

مهندس أسامة الحسيني

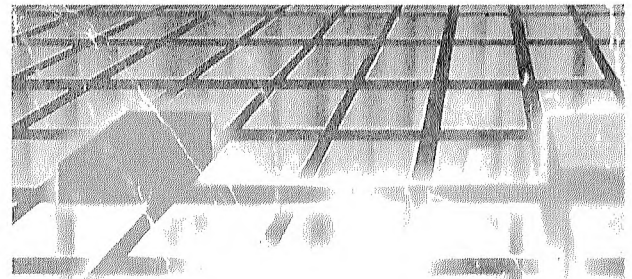
مصمم نظم بشرية تكنولوجية

انترناشيونال

الولايات المتحدة الأمريكية

# تحدث إلى الكمبيوتر بـ لغة

# PASCAL



2010/10/10



مهندس أسامة الحسيني

مصمم نظم بشركته تكنولوجي

انترناشيونال

الولايات المتحدة الأمريكية

# تحدث إلى الكمبيوتر

بلغة

PASCAL پاسكال

مكتبة الساعي

الرياض تليفون ٤٢١٥٦٣٦ - ٤٢١١٤٢٤

ص. ب. ٥٠٦٤٩ - الرياض ١١٥٣٣

## مكتبة ابن سينا

نافذتك على الفكر العربي  
والعالمى بما تقدمه لك من روائع  
الكتب العلمية والفنية والتراثية  
التي تجمع بين الأصالة والمعاصرة.

يديرها ويشرف عليها  
مهندس / مصطفى عاشور

جميع الحقوق محفوظة للناسـر



بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



# بسم الله الرحمن الرحيم

كلمة المؤلف

## باحث عن الدهشة في بلاد العم سام !

حملت عصاى وشددت الرحال إلى بلاد العم سام فكان أول ما بحثت عنه « أسرة الكمبيوتر » التى كنت أتمنى إليها فى مصر وفى مدينتى الاسكندرية لكن حقيقة هائلة أذهلتنى فى النهاية وهى أن الناس لا يحبون الكمبيوتر بنفس الكيفية التى نحبها فى مصر . نعم هم يستخدمون الكمبيوتر ويعتمدون عليه فى العمل بصفة مطلقة . لكننى لم ألتق بشاب يحدثنى عن الكمبيوتر أو ناشئ يعشق البرمجة أو طفل يتعجل الخطى نحو التقدم فى التعامل مع الكمبيوتر ..

كانت أسرة الكمبيوتر فى مصر تضم الشباب والنشء والأطفال وأساتذة الجامعات يربطهم جميعاً حب الكمبيوتر والاهتمام بكل ما يخصه من أخبار . حتى كان الحديث عن الكمبيوتر هو حديث المقهى والنادى وحديث الطريق العابر . وكم من أطفال فى مصر برعوا فى برمجة الكمبيوتر بدرجة تفوق أعمارهم ومستوى دراستهم .

أما الطفل الأمريكى فبرغم أنه يدرس مادة الكمبيوتر كمادة أساسية فى المدرسة لكنه ينظر إلى الكمبيوتر كما ينظر إلى شجرة فى الطريق . لا يدهشه الروبوت أو الكمبيوتر الذى يأتى بالأعاجيب .

ولعل السبب فى ذلك يرجع إلى أن الرجل الأمريكى لا يرى فى الكمبيوتر إلا وسيلة لإدارة أعماله وتسهيل مصالحه . فالحضارة الأمريكية لم تنشأ إلا نتيجة الاحتياج والبحث عن الحلول . والرجل الأمريكى لا يهتم أن يجيد لغة

خلاف اللغة الانجليزية لأنه لا يحتاج إليها .

وعندما تحولت باحثاً عن المحترفين ، عثرت أخيراً على « دراويش » الكمبيوتر لكنني وجدت السواد الأعظم منهم من الأجانب الذين جاءوا من مشارق الأرض ومغاربها سعياً وراء البحث العلمى والتطوير ، أو لعل الدهشة هى التى جذبت الكثيرين منهم إلى أضواء الحضارة الأمريكية .  
وهمس فى أذنى أحد الأصدقاء قائلاً :

« إن الأمريكيين أنفسهم لا ينكرون أن حضارتهم قامت على أكتاف هؤلاء الأجانب الباحثين عن المغامرة والدهشة ! » ولعل هذه الكلمات جعلت منطق الأشياء يستقيم فى ذهنى من جديد ، فقد اعتقدت دائماً أن الدهشة هى البداية الحقة للإبداع . وأن الطفل المصرى الذى يستولى عليه « هوس » الكمبيوتر ويجعله ينفق آخر مليم فى جيبه لكى يقتنى برنامجاً أو كتاباً عن الكمبيوتر ، هذا الطفل بعينه سوف يكبر ذات يوم ويأتى منه ما أتى من جهازة الباحثين الذين تدين لهم البشرية بالفضل والعرفان .

أسامة الحسينى



## الباب الأول

القواعد العامة للغة باسكال



## مفتح

لعلها فكرة صائبة أن نبدأ بالتعميم ثم ننتقل إلى التخصيص فنبداً بعرض الصورة الكاملة لبرنامج باسكال بهدف التعرف على شكله العام وخصائصه ثم نتدرج بعد ذلك في تقديم المكونات الأساسية التي بنى منها البرنامج : الكلمات والعبارات والمؤثرات والتعبيرات !

ولعل هذه الطريقة قد تنجح أكثر في جذب القارئ الجديد على لغات الكمبيوتر الذى يتعجل أن يرى البرنامج المتكامل ولو كان برنامجاً صغيراً ، ولعله أيضاً يتعجل الجلوس إلى جهاز الكمبيوتر لتجربة البرامج ومشاهدة النتائج .

أما القارئ المتمرس في اللغات الأخرى فلا أعتقد أنه سيشعر بالملل معنا وهو يتابع البرامج التمهيدية فسرعان ما نصل إلى الثوابت والمتغيرات وقواعد اللغة بكل تقعراتها !

## (١ - ١) لغة باسكال :

تتميز لغة باسكال ببنائها المتفرد . وأولى ملامح هذا البناء أن البرنامج المكتوب بلغة باسكال سهل الكتابة ، سهل القراءة لكونه مبنياً بناءً طبيعياً متسلسلاً . أما الخاصية الثانية فهي الطريقة المتميزة التي تبنى بها البيانات والتي تعطى المبرمج درجات حرية في استخدام هذه البيانات . وبذلك فإن برنامج باسكال يتميز بصفة عامة بالوضوح وهي الصفة التي يجب أن تتميز بها اللغات عالية المستوى (high level languages) لتسهيل كتابة البرامج وإصلاحها .

وبقدر ما هي قوة في أدائها للأغراض المختلفة بقدر ما هي اقتصادية من ناحية الزمن المستغرق لتنفيذ البرامج وكذلك من ناحية السعة التي يشغلها البرنامج .

وقد أثبتت لغة باسكال فعاليتها — مع ظهور الميكرو كومبيوتر ، ومع ذلك فهي تستخدم في الأجهزة الكبيرة (Mainframe) والمتوسطة (Mini) والصغيرة (Micro) بفوارق بسيطة بين الطرازات .

وقد سميت لغة باسكال بإسم العالم الفرنسي بليز باسكال BLAISE PASCAL (١٦٦٢ — ١٦٩٣) الذي ابتكر آلة حاسبة ميكانيكية في وقت مبكر . أما اللغة نفسها فقد قدمها الدكتور نيكولوس فيرت Niklaus Wirth رئيس قسم علوم الكمبيوتر في معهد التكنولوجيا الفيدرالية بزيورخ — عام ١٩٧٠ .

## (١ - ٢) ترجمة اللغات عالية المستوى :

تتميز اللغات عالية المستوى بأنها تقدم التسهيلات لحل المشكلة المعينة موضوع البرنامج بصرف النظر عن تفصيلات لغة الماكينة الثنائية (binary code) ولذلك فهي تصلح للمستخدم العادي الذي لا يريد أن يشغل نفسه كثيراً بما في داخل الكمبيوتر !



وبالطبع فإن أجهزة الكمبيوتر لا تفهم اللغات عالية المستوى وتحتاج دائماً  
لمترجم والترجمة عادة تكون بأحد وسيلتين :

( أ ) الترجمة التجميعية **COMPILATION**

(ب) الترجمة الفورية **INTERPRETATION**

والترجمة الفورية تتم سطراً بسطر (أو أمراً بأمر) كما في لغة بيسك (BASIC)  
وهذا يتم بواسطة برنامج آخر يسمى المترجم الفوري (interpreter) يراجع  
معك البرنامج كلما أدخلت منه سطراً للكمبيوتر ويصحح لك الأخطاء .

أما الترجمة التجميعية فتتم جملة واحدة للبرنامج ككل ويقوم بها المترجم  
التجميعي (COMPILER) وبعد الترجمة تحصل على نسخة جديدة من البرنامج  
مكتوبة بلغة الماكينة . أما إذا كان البرنامج محتويّاً على أخطاء فإنك تحصل على  
تقرير بهذه الأخطاء ، وفي هذه الحالة فإنه يلزم إعادة ترجمة البرنامج بعد إصلاح  
الأخطاء .

ويسمى البرنامج المكتوب باللغة عالية المستوى « البرنامج المصدر » (source  
program) ، وأما البرنامج المكتوب بلغة الماكينة والنتائج من عملية الترجمة  
فليسعى : البرنامج الهدف (object program) .

### ( ١ - ٣ ) ترجمة لغة باسكال :

من قرأ كتبنا عن لغة فورتران أو لغة كوبول سوف يلاحظ بلا شك أننا  
خصصنا جانباً كبيراً من المناقشة لعملية الترجمة . ولكنه في الحقيقة أن اللغة  
كلما زاد انتشارها وتعددت طرازاتها كلما أصبح من غير المجدي دراسة مترجم  
بعينه . ففي هذا الكتاب نهتم أكثر بعرض فنون البرمجة بلغة باسكال بصرف  
النظر عن طراز اللغة المستخدم وبصرف النظر عن نوع الكمبيوتر الذي تعمل  
عليه . أما خطوات الترجمة فهي عادة تكون مشروحة في كتاب اللغة الذي  
تشتريه مع مترجم لغة باسكال المحفوظ على القرص المغنطيسي أو الشريط

المغناطيسي . وبعض طرازات لغة باسكال تحتوي على كل من الترجمة التجميعية والترجمة الفورية .

كما أنه مع أجهزة الكمبيوتر الشخصي: IBM والأجهزة المتوافقة معه ، نجد أنواعاً متقدمة من برامج الترجمة توفر على المستخدم الجهد المبذول في ترجمة البرنامج وتعفيه من هذا كله بإعطاء أمر واحد فقط ، هو أمر التنفيذ ! ويقوم البرنامج المترجم تلقائياً بترجمة وتنفيذ البرنامج . وأحد أمثلة هذا المترجم هو المترجم السريع TURBO PASCAL .

ويمدك المترجم السريع بمحرر خاص (editor) لتكتب عليه سطور البرنامج وبه كل الوسائل التي تسهل عملية تحرير البرنامج من كتابة وحذف وإدماج للكلمات أو حفظ واستدعاء للبرامج والملفات . ويتم عملية الترجمة بالضغط على زر واحد هو الحرف C (اختصار Compilation) فتحصل على تقرير كامل بتفصيلات عملية الترجمة ورسائل خاصة عن الأخطاء الموجودة بالبرنامج . كما يمكن إعطاء أمر التنفيذ مباشرة بالضغط على الحرف R (اختصار Run) فتم عملية الترجمة والتنفيذ متتابعتين إذا كان البرنامج خالياً من الأخطاء وإلا فتحصل على تقرير الترجمة ورسائل الأخطاء .

والشكل (١ - ١) يوضح برنامجاً صغيراً بلغة باسكال مكتوب على صفحة المحرر ونرى أعلى الصفحة اسم البرنامج وعدّاد السطور وعدّاد الأعمدة . كما يوضح شكل (١ - ٢) التقرير الناتج من عملية الترجمة يعقبه تنفيذ البرنامج .

Line 9 Col 1 Insert Indent A:5123MOD.PAS  
 عدد السطور عدد الأعمدة اسم الموضع المعطى اسم البرنامج

```

program chararrays(input,output);
VAR ch: array[1..20] of char;
    n,i: integer;

BEGIN
lowvideo;

    readln(n);
    for i:=1 to n do
        read(ch[i]);

highvideo;

    writeln;
    for i:=1 to n do
        writeln(ch[i]);
END.
  
```

سطور البرنامج تنال  
 على صفحات محرر لغة باسكال

## شكل (١ - ١)

برنامج باسكال على أحد صفحات المحرر (editor)

>

Compiling ← عملية الترجمة  
 18 lines

Code: 000C paragraphs \ 192 bytes, 0D1C paragraphs tree  
 Data: 0004 paragraphs \ 64 bytes, 0FD8 paragraphs tree  
 Stack/Heap: 8944 paragraphs \ 50376 bytes

Running ← عملية التنفيذ  
 5  
 abcde  
 a  
 b  
 c  
 d  
 e

البيانات المنطوقة  
 المعلومات الخارجة  
 بعد المعالجة

>

## شكل (٢ - ١)

تقرير المترجم وتنفيذ برنامج باسكال

## (١ - ٤) نظرة شاملة إلى برنامج باسكال :

قبل أن نغرق في التفاصيل ، دعنا نلقى نظرة شاملة على البرنامج المكتوب بلغة باسكال حتى نتعرف إلى ملامحه العامة ولنبدأ بهذا البرنامج الصغير شكل (٣-١) :

```

program demo(output);
begin
    writeln('HI THERE..');
    writeln('MY NAME IS HAZEM OSSAMA AL-HUSSEINY');
    writeln('IT'S BEEN NICE TO MEET YOU');
    writeln('KEEP IN TOUCH...');
end.

```

البرنامج

اسم البرنامج

البداية

النهاية

عمليات المعالجة

### شكل (١ - ٣)

أياً كان هدف البرنامج فسوف نراه دائماً في هذا الهيكل العام حيث يبدأ بالسطر الذي يحتوي على اسم البرنامج . ثم تتالى عمليات المعالجة المختلفة بين البداية (begin) والنهاية (end.) وهذا البرنامج يقوم بطبع أربعة سطور متتالية على الشاشة كما هو موضح في الشكل (١ - ٤) :

```

HI THERE..
MY NAME IS HAZEM OSSAMA AL-HUSSEINY
IT'S BEEN NICE TO MEET YOU
KEEP IN TOUCH...

```

تنفيذ البرنامج

### شكل (١ - ٤)

وأول ما يلفت النظر عندما نلقى نظرة أولى على البرنامج ، هو هذه العبارات التي تحتها خط . وبالطبع فإن هذه الخطوط لا تعتبر ضمن البرنامج ولا تكتب العبارات بهذه الصورة ، وإنما قد وضعناها لإبراز بعض الكلمات المحجوزة

التي يختص بها برنامج باسكال والتي لا يجوز استخدامها في غير أماكنها بالبرنامج .

ولنستهل قراءة البرنامج بالسطر الأول لكي نَدون بعض الملاحظات :

### **program demo(output);**

يبدأ البرنامج بالكلمة المحجوزة program يعقبها اسم البرنامج demo الذي نختاره على هوانا . أما الكلمة output التي جاءت بين القوسين فهي تخبر الكمبيوتر بأن يستعد لطباعة بعض العبارات عندما نعطي أمر التنفيذ . وقد نلتقي بكلمة input في نفس هذا المكان مع برامج أخرى ، وهذه الكلمة تخبر الكمبيوتر بأن يستعد لاستقبال بعض البيانات من العالم الخارجي (من مستخدم البرنامج) !

فإذا كان البرنامج يستقبل البيانات ويطبع المعلومات معاً سوف نجد كلمة input, output بين القوسين . والعبارة program وما يتبعها عبارة اختيارية يمكن الاستغناء عنها في الكثير من الطرازات . فإذا انتقلنا إلى السطر الثاني من البرنامج نجد كلمة (begin) بمعنى « ابدأ » . وهي دائماً تكتب قبل عمليات المعالجة المختلفة التي قد يتضمنها البرنامج مثل الطباعة وقراءة البيانات وإجراء العمليات الحسابية إلى آخره . وينتهي البرنامج بالكلمة (end.) مصحوبة بنقطة النهاية .

أما عمليات المعالجة نفسها فقد كانت كلها عمليات طباعة باستخدام العبارة (writeln) . وقد جاءت المعلومات المطلوب طباعها بين علامتي اقتباس مثل :

### **writeln('HI THERE..');**

وعلامة الاقتباس المستخدمة في لغة باسكال هي علامة الاقتباس المفردة ، وهي نفسها العلامة apostrophe المستخدمة في اللغة العادية . ولذلك إذا

كانت الجملة (الرسالة) المطلوب طباعتها تحتوى على علامة اقتباس كما فى السطر الثالث :

## IT'S BEEN NICE TO MEET YOU

فإن هذا يتطلب استخدام علامتى اقتباس متتاليتين كما فى البرنامج .

('IT'S BEEN NICE TO MEET YOU');

ولعله يبدو ملفتاً للنظر استخدام الأقواس علاوة على علامات الاقتباس مع العبارة writeln .

أليس هذا بالكثير ؟ ...

هناك سبب لذلك . وهو أن عبارة الطبع لها استخدامات أخرى ، فهى قد تستخدم لطبع الرسائل (messages) أو الحرفيات (strings) كما جاء فى هذا المثال ، وقد تستخدم لطبع الأعداد ، كما تستخدم أيضاً لإجراء بعض العمليات الحسابية قبل طباعة النتائج . وهذا يقدمنا لمثال آخر من برامج باسكال شكل (١ - ٥) :

```
program room(output);
```

```
begin
```

```
  writeln('your room needs ', 3*2, ' sq.m. of vinyl');
  writeln('and ', 2*(3+2)*3, ' sq.m. of wallcovering.')
```

```
end.
```

## شكل (١ - ٥)

يقوم هذا البرنامج بحساب مساحتى الأرضية والحوائط لحجرة ذات أبعاد معينة بهدف تغطيتها « بالفينيل » وورق الحائط فإذا دخلنا إلى صلب البرنامج مباشرة فإننا نلتقى بالعبارة writeln الأولى والتي تحتوى بين قوسها على ثلاثة

أجزاء مختلفة وقد استخدمت علامة الفاصلة (comma) للفصل بين الأجزاء .

الجزء الأول هو :

### **your room needs**

نلاحظ وجود هذه العبارة بين علامتى اقتباس وهذا يعنى أن كل المطلوب من الكمبيوتر هو طبع ما بين العلامتين دون أدنى تصرف .

فلنر الجزء الثانى :

**3\*2**

جاء هذا الجزء بلا علامات اقتباس . معنى هذا أنه على الكمبيوتر أن يجرى عملية معالجة معينة قبل أن يطبع النتيجة . والمعالجة المطلوبة هنا هى ضرب العددين 3 ، 2 ثم طبع الناتج .

أما الجزء الثالث فقد أتى بين علامتى اقتباس لذلك سيطبع كما هو .

### **sq.m. of vinyl**

لذلك نتوقع أن تكون نتيجة تنفيذ هذا السطر هى طباعة السطر التالى من المعلومات .

### **your room needs 6 sq.m. of vinyl**

يقول هذا السطر إن حجرتك تحتاج إلى ٦ أمتار مربعة من « الفينيل » ! ربما يضيف الكمبيوتر بعض المسافات الخالية (spaces) قبل الرقم 6 بحسب طراز اللغة ولكن هذه هى النتيجة المتوقعة بصفة عامة . وكما نرى أن النتيجة كلها جاءت على سطر واحد وهذا هو تأثير العبارة writeIn حيث يدل الحرفان In على وحب الانتقال إلى سطر جديد بعد طبع المطلوب .

عند تنفيذ العبارة التالية (السطر الرابع) سيقوم الكمبيوتر بطباعة السطر التالي :

### **and 30 sq.m. of wallcovering**

وفي هذا السطر طبع الكمبيوتر الجزئين بين علامتى الاقتباس بدون تصرف . أما العملية الحسابية فقد حسب نتيجهها فكانت 30 .  
وتقول هذه الفقرة : إن الحجرة تحتاج أيضاً إلى ٣٠ متراً مربعاً من ورق الحائط .

### ● الأخطاء النحوية Syntax errors :

نلاحظ في هذا البرنامج ضرورة استخدام الفواصل (commas) والفواصل المنقوطة (semicolons) في الأماكن المحددة لها بالضبط ، وإلا فإن الكمبيوتر يرسل لك رسالة إنذار بوقوع خطأ ما في قواعد اللغة (syntax error) وبالطبع يتوقف الكمبيوتر عن التنفيذ إذا صادف خطأ مثل هذا (عادة تكتشف مثل هذه الأخطاء أثناء الترجمة) .

أما طريقة تنظيم البرنامج فلا تخضع لقواعد مشددة فيمكننا أن نترك ما نشاء من المسافات الخالية بين الكلمات والأرقام والرموز .  
ولكن القاعدة هنا أن تكون هناك مسافة واحدة على الأقل بين الكلمات .  
كما أن لنا الحرية في تنظيم سطور البرنامج بالطريقة التي تسهل علينا قراءته ومراجعته .

### ● أخطاء التنفيذ Execution errors :

ربما يكون البرنامج سليماً من ناحية قواعد اللغة وخالياً تماماً من الأخطاء النحوية ، لكن الكمبيوتر مع ذلك يتوقف عن التنفيذ مرسلًا لك رسالة أخرى عن سبب عدم استطاعته تنفيذ البرنامج .



قد يكون السبب هو وجود عدد ما في الحسابات الناتجة لا يستطيع التعامل معه لكونه كبير جداً أكبر من حدود الكمبيوتر . أو قد يكون السبب وجود عملية حسابية معضلة مثل القسمة على صفر وهى تخرج عن حدود أى كمبيوتر .

مثل هذه الأخطاء التى تحدث أثناء التنفيذ تسمى أخطاء التنفيذ (execution errors) .

### ● الأخطاء المنطقية Logical errors :

بعد أن يتأكد المبرمج من خلو البرنامج تماماً من الأخطاء النحوية وأخطاء التنفيذ المحتملة يعطى الأمر بتنفيذ البرنامج فيقوم الكمبيوتر بطبع النتيجة على الورق أو على الشاشة فى ثوان فيسعد المبرمج نفساً ! ولكن نظرة واحدة إلى النتيجة تصيبه بخيبة أمل إذ يكتشف أن الكمبيوتر قد أجرى حسابات خاطئة وطبع نتيجة غير معقولة .. غير منطقية .. فماذا حدث ؟ .

بالطبع الذنب ليس ذنب الكمبيوتر فهو يفعل ما يؤمر به طالما خاطبناه باللغة الصحيحة التى يفهمها وفى حدود إمكانات أجهزته . هذه النوعية من الأخطاء تسمى الأخطاء المنطقية ولا يستطيع الكمبيوتر التعرف عليها ولذلك لا يتوقف عندها .

فمثلاً لو أن المبرمج قد استبدل علامات الجمع (+) بعلامة الطرح (-) فى التعبير الحسائى الوارد فى السطر الرابع أى أنه كتب التعبير (سهواً) بالصورة الآتية :

$$2 * (3 - 2) * 3$$

فإن نتيجة هذه العملية الحسابية سوف تكون الرقم 6 بدلاً من النتيجة التى كان يقصدها المبرمج وهى العدد 30 . وهذا بالطبع ليس ذنب الكمبيوتر فهو لا يعلم قصد المبرمج إذا هو أخطأ التعبير .

## ● جدوى البرنامج :

لا شك أن القارئ الجديد على لغات الكمبيوتر سوف يجد هذا البرنامج نافهاً في هدفه ولعله يتساءل عن جدوى هذا البرنامج الذى نبذل جهداً فى إعدادده لكى يخرج علينا فى النهاية بنتيجة عملية حسابية سهلة كان من الممكن أن نحسبها شفهاً دون الحاجة إلى الكمبيوتر « من أصله » !

وهذا حق ... ولكن مهلاً !

فحتى هذا البرنامج الصغير يمكنه أن يؤدى عملاً نافهاً لو أنه كان يؤدى نفس العملية الحسابية لختلف الحجرات ذات الأبعاد المختلفة ...

وهذا يمكن أن يتحقق إذا جعلنا البرنامج يستقبل بيانات جديدة فى كل مرة ويخترنها فى متغيرات مثل طول الحجرة وعرض الحجرة وارتفاعها .

فلا يمكن أن نتصور بحال أن ننشئ برنامجاً لحساب مرتب شخص واحد ، فالبرنامج عادة يكون برنامجاً عاماً يحتوى على متغيرات مثلاً الاسم وعدد ساعات العمل وعدد الساعات الإضافية إلى آخره .. ويقوم البرنامج باستقبال القيم العددية للبيانات الخاصة بكل موظف ويحسب له صافى الأجر على حدة .

وفى برنامجنا الصغير يمكننا أن نمثل الطول بالمتغير length والعرض بالمتغير width والارتفاع بالمتغير height فيصبح البرنامج كما فى شكل (١ — ٦) :

```

program roomsize2(input,output);
var length, width, height : integer;
begin
    writeln('type in length, width and height');
    read(length, width, height);
    writeln('your room needs ', length*width,
            ' sq.m. of vinyl');
    writeln('and ', 2*(length + width)*height,
            ' sq.m. of wall covering');
end.

```

المتغيرات

ادخال قيمة المتغيرات الثلاثة

### شكل (١ - ٦)

عند تشغيل مثل هذا البرنامج سوف يسألنا أن ندخل طول وارتفاع وعرض الحجرة المقصودة فيقوم على الفور بحساب المساحات اللازمة من ورق الحائط وغطاء الأرضية وذلك بفضل العبارة **read** التي ظهرت في السطر الثاني .

وقد ظهر بهذا البرنامج سطر جديد قبل بداية البرنامج مباشرة بدأ بكلمة **VAR** ، وفائدة هذا السطر هو الإعلان عن المتغيرات المستخدمة في البرنامج . وسوف نشرح هذا الموضوع تفصيلاً في الأجزاء القادمة .

بقيت ملاحظة هامة يجب علينا أن نذكرها ، فلا شك أن بعض الأصدقاء الجدد على لغة باسكال ينساءون « هل من الضروري أن نكتب برنامج باسكال بالحروف الصغيرة ؟ » في الحقيقة أن هذه النقطة قد تختلف فيها أجهزة الكمبيوتر وطرزات اللغة المستخدمة . فمع بعض الطرازات قد يستلزم الأمر أن تكتب بعض الكلمات المحجوزة باستخدام الحروف الكبيرة (Capital or upper case) أما بقية البرنامج فيكتب بالحروف الصغيرة (Small or lower case) . ومع ذلك فالأجهزة الحديثة والطرزات الحديثة من لغة باسكال أصبحت تتغاضى عن هذا كله وتتسامح في مثل هذه الشكليات ،

فالبرنامج التالي منفذ على أحد أجهزة الكمبيوتر المتوافقة مع IBM ومكتوب بالحروف الكبيرة شكل (١ - ٧) وهو نفس البرنامج الذي عرضناه من قبل .

PROGRAM ROOM(OUTPUT);

BEGIN

WRITELN('YOUR ROOM NEEDS -', 3\*2, ' SQ.M. OF VINYL');

WRITELN('AND ', 2\*(3+2)\*3, ' SQ.M. OF WALLCOVERING.')

END.

### شكل (١ - ٧)

وبالطبع يخرج عن المناقشة ما جاء بين علامتي الاقتباس مع عبارة الطباعة writeln فالحرفيات قد تكتب بأى صورة تراها .

أما القاعدة التى لا يتساعح فيها مترجم باسكال فهى ضرورة فصل العبارات عن بعضها البعض بفاصلة منقوطة (semicolon) . لأن كل عبارة تمثل أمراً مستقلاً ينفذ على حدة . ولعلك تلاحظ أن الفاصلة المنقوطة لم توضع بعد العبارة الأخيرة لأنها لا داعى لها .

### ● القواعد العامة لبرنامج باسكال :

لا توجد قواعد كثيرة تحدد شكل البرنامج المكتوب بلغة باسكال مثل الكتابة على نموذج معين أو فى مكان معين من السطر ، بل يكفى أن تبدأ العبارة من أول السطر . وحتى يكون بناء البرنامج واضحاً يفضل أن تبدأ العبارات من نفس المسافة دائماً .

وتُفصل كلمات باسكال بمسافة خالية أو أكثر أو برمز خاص أو بنهاية السطر .

ويحتوى البرنامج بصفة عامة على الحروف الأبجدية والأرقام والعلامات الخاصة وأى منها يسمى لبنة (character) ، ومن اللبئات تتكون الكلمات

(words) ومن الكلمات تبني العبارات (statements) التي تمثل وحدات بناء برنامج باسكال .

## (١ - ٥) أسماء البيانات (Identifiers) :

إن أسماء البيانات هي أسماء الأوعية التي توضع فيها البيانات سواء كانت هذه البيانات أرقاماً مثل « الأجر » أو حرفيات مثل « الاسم » و « رقم التليفون » ... إلى آخره . ومن درس لغة يسك من قبل فقد عرف أسماء البيانات تحت اسم المتغيرات (variables) ، أما في لغة كوبرول فهي تسمى (data-names) . وها هي أسماء البيانات تهل علينا تحت اسم جديد هذه المرة وهو (identifiers) مع فارق واحد وهو أن أسماء البيانات في لغة باسكال تستخدم لتسمية المتغيرات علاوة على طراز جديد من البيانات وهو الثوابت المسماة كما سنعرف في الفقرات التالية .

والقواعد العامة التي تخضع لها أسماء البيانات هي :

( أ ) يتكون اسم البيان من الحروف الأبجدية (من A إلى Z) أو من الأرقام (من 0 إلى 9) أو من كليهما بشرط أن يبدأ الاسم دائماً بحرف مثل :

**netpay , telephone , x1**

أما بالنسبة لاستخدام الحروف الصغيرة أو الكبيرة مع أسماء البيانات فإن طرازات اللغة تختلف في ذلك كما ذكرنا من قبل ولذلك يجب الاسترشاد في هذه النقطة بدفتر اللغة الخاص بالكمبيوتر المعين .

(ب) لا يجب أن يتضمن الاسم مسافات خالية .

(ج) بالنسبة لطول اسم البيان ، فداًئماً هناك حد أقصى (ويكون ٣٠ لبنة مع أغلب الطرازات) .

ومع ذلك فإن مترجم لغة باسكال لا يلتفت إلا إلى الحروف الثمانية الأولى

من الاسم .

فإذا استخدمت اسمين مثل :

**temperate temperature**

فإنهما يصبحان اسماً واحداً عند ترجمة البرنامج .

وبصفة عامة تفضل الأسماء الطويلة التي تشرح معناها ذاتياً بدلاً من الحروف المفردة .

### ( ١ - ٦ ) الكلمات المحجوزة (Reserved Words) :

تتميز لغة باسكال بقلّة الكلمات المحجوزة (المحظور استخدامها كأسماء للبيانات) ، وهذه الكلمات تتميز بأنها تحمل معنىً معيناً يفهمه الكومبيوتر على نحو ما مثل :

**PROGRAM , IF , OR , THEN .**

والملاحق ( أ ) في نهاية الكتاب يتضمن قائمة بالكلمات المحجوزة في لغة باسكال .

### ( ١ - ٧ ) الكلمات القياسية (Standard Words) :

هناك مجموعة أخرى من الكلمات الخاصة السابق تعريفها للمترجم ، ورغم أنه يمكن استخدامها كأسماء للبيانات لكن هذا يفسد معناها الأصلي الذي ابتكرت من أجله لذلك لا يُنصح باستخدامها . وسوف يلي الحديث عن فائدتها في الفقرات القادمة . والملاحق (ب) يضم هذه النوعية من الكلمات .

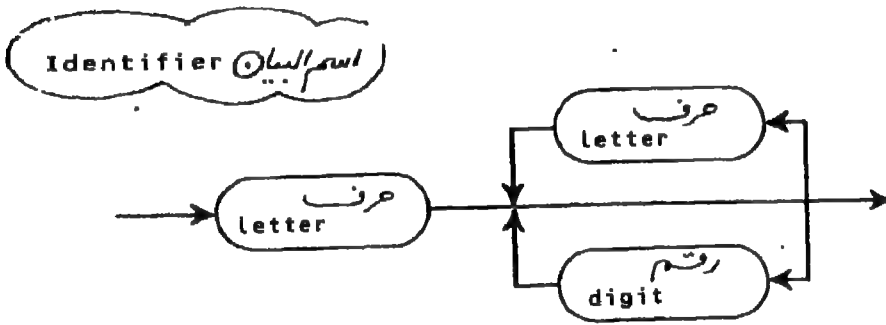
### ( ١ - ٨ ) خرائط قواعد اللغة (Syntax Diagrams) :

عادة تصاغ قواعد لغة باسكال بواسطة الخرائط وهى طريقة أكثر تعبيراً

وأسرع استقبلاً من القارىء .

وتستخدم فى الخرائط الحروف الكبيرة وعلامات الترقيم للتعبير عن أشياء قائمة بذاتها (مثل الكلمات المحجوزة وعلامات العمليات الحسابية) ، وتستخدم الحروف الصغيرة لتدل على المكونات الأخرى التى تتغير حسب الموقف مثل أسماء البيانات والثوابت .. وهذا مجرد عرف لكنه ليس ملزماً .

وشكل (١ - ٨) يوضح خريطة قاعدة أسماء البيانات السابق شرحها .



شكل (١ - ٨)

(٩ - ١) التعليقات Remarks :

من الممكن إضافة أية ملاحظات أو تعليقات للبرنامج باستخدام علامة خاصة فتبدأ الملاحظة بالقوس الأيسر { وتنتهى بالقوس الأيمن } ووجود هذه العلامة الخاصة يجعل الكمبيوتر لا يلتفت لما بعدها من ملاحظات حتى القوس

الأمين الذى تكتمل به الملاحظة .

وكتابة التعليقات فى البرنامج له أهمية كبيرة لا سيما عندما ترغب فى تعديل برنامج مضى وقت طويل على كتابته فإنه بالاستعانة بالملاحظات المكتوبة داخل البرنامج يمكنك تذكر بعض الخطوات بسرعة . كما أنها تفيد فى النواحي التعليمية .

وفى أجهزة الكمبيوتر التى ليس بها هذا النوع من الأقواس يمكنك استخدام علامتين (\*) بدلاً من القوس الأيسر والعلامتين (\*) بدلاً من القوس الأيمن .

ولو كانت الحروف العربية متوفرة فى جهاز الكمبيوتر فلا بأس من كتابة التعليقات باللغة العربية .





## ■ تمارين على الباب الأول :

س (١ - ١) أذكر أى من أسماء البيانات التالية جائز وأيها غير جائز مع بيان السبب :

- |               |                 |
|---------------|-----------------|
| (a) a         | (f) VAR         |
| (b) datafile1 | (g) code number |
| (c) xrolp5z   | (h) real        |
| (d) xi/9      | (i) datafile2   |
| (e) passmark  | (j) 7 - up      |

س (١ - ٢) فى المثال السابق كلمات جائزة كأسماء للبيانات ولكنها لا توصى بها . استخرج هذه الكلمات وعلّق عليها من وجهة نظرك .





## الباب الثاني

# الأنماط والتعبيرات Types & Expressions



## مفتح

تندرج الأدوات المستخدمة في البرنامج (البيانات والمتغيرات والثوابت المسماة) تحت نوعيات مختلفة هي الأنماط . ومن مستلزمات برنامج باسكال هي الأنماط . ومن مستلزمات برنامج باسكال أن نعلن في بدايته عن أنماط كل ما نريد استخدامه من أدوات حتى يعد الإجراءات المناسبة للتعامل مع هذه الأدوات وفقاً لأنماطها المختلفة .

ومن الثوابت والمتغيرات والثوابت المسماة تتألف التعبيرات التي تندرج بدورها تحت أنماط مختلفة .

وفي هذا الباب نقدم لك كيفية تنظيم أدواتك المستخدمة في البرنامج وكيفية بناء التعبيرات المختلفة باستخدام هذه الأدوات .

## (٢ — ١) الأنماط القياسية (Standard Types) :

ولا تنتمي البيانات جميعاً إلى نوعية واحدة فالأسماء تختلف عن الأعداد والأعداد منها الصحيح ومنها الكسر . والكومبيوتر يتعامل مع كل نوعية بطريقة مختلفة لذلك فإن لغة باسكال تحتوى على نوعيات قياسية من البيانات وأسمائها يطلق عليها الأنماط القياسية (Standard types) . وبجانب ذلك فإن اللغة تسمح أيضاً بابتكار أنماط خاصة بالمبرمج كما سنرى في الفصول المتقدمة للكتاب . أما الأنماط القياسية فهي أربعة :

### (١) النمط الصحيح (integer) :

وهو النمط الذى تنتمي إليه الأعداد الصحيحة التى لا تحتوى على كسور أو علامة عشرية .

### (٢) النمط الحقيقى (real) :

وهو النمط الذى تنتمي إليه الأعداد المحتوية على علامة عشرية أو كسور عشرية . -

### (٣) نمط اللبنة (char) :

وينتمى إلى هذا النمط كل اللبئات التى تحتوى عليها لغة باسكال وهى الحروف الأبجدية والأرقام من 0 إلى 9 والعلامات الخاصة بأنواعها .

### (٤) الأنماط المنطقية (البوليانية) (Boolean) :

وهى تمثل حالتى المنطق :

(true)

صحيح

(false)

أو غير صحيح

## (٢ - ٢) الثوابت (Constants) :

الثوابت هي البيانات التي نتعامل معها مثل أسماء الأشخاص أو الأعداد المألة على الأجر والمخزون والرصيد أو أرقام التليفونات وهي تدرج تحت أحد الأنماط الآتية :

### (١) الثوابت الصحيحة (Integer constants) :

مثل الأعداد : 1281 ، -22 ، 125  
واستخدام علامة الموجب مع الثوابت اختياري .

### (٢) الثوابت الحقيقية (Real constants) :

وهي قد تكتب بالطريقة المعتادة مثل :  
4.0 ، 0.233 ، -22.5  
أو قد تكتب باستخدام الطريقة الأسية مثل :  
4.5E+6 ، 5.2E-7

ملاحظة :

يعتبر العدد  $5.2E-7$  مساوياً للعدد  $5.2 \times 10^{-7}$  .  
كما يعتبر العدد  $4.5E-6$  مساوياً للعدد  $4.5 \times 10^6$  .

### (٣) ثوابت اللبئات (Char constants) :

مثل 'A' ، 'B' ، '?'

### (٤) الثوابت المنطقية (Boolean constants) :

وهي rue و else

### (٥) ثوابت الحرفيات (string constants) :

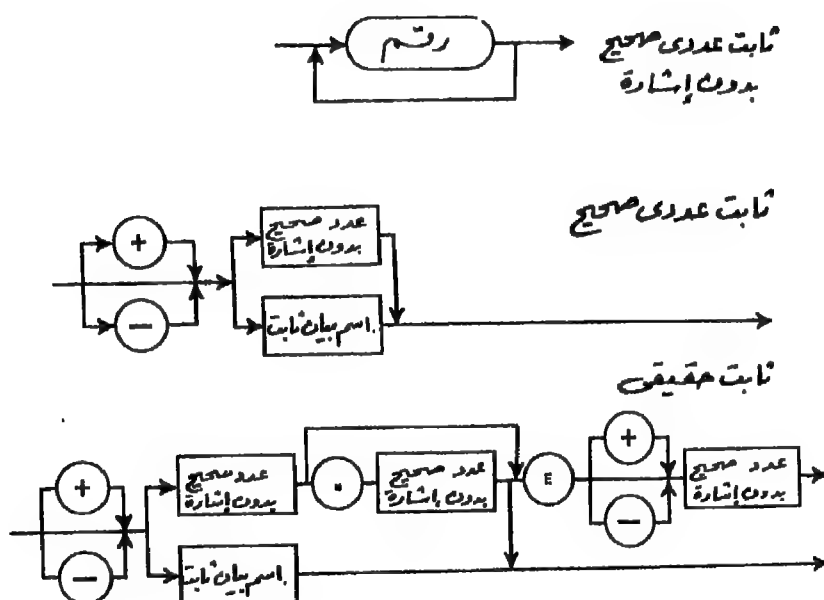
وعلاوة على الأنماط السابقة (القياسية) فإنه يمكن ضم مجموعة من اللبئات

معاً لتكوين ثابت حرفي (string) وقد يطلق عليها أيضاً مصفوفة اللبنة  
المُحزّمة (وسياقي الحديث عنها في الأجزاء المتقدمة) .

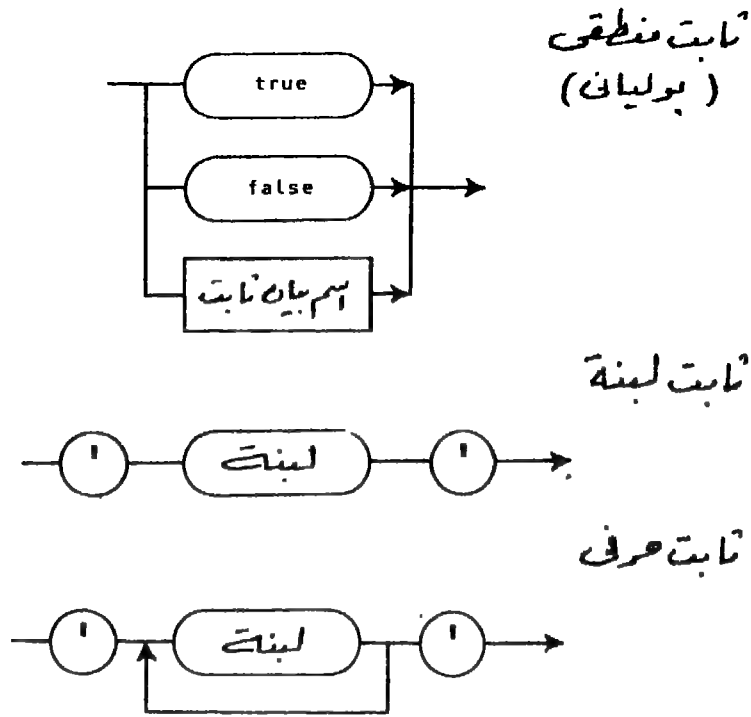
وأمثلة ثوابت الحرفيات هي :

**'ABC' , 'Alexandria' , 'Hazem'**

والشكل التالي (٢ - ١) يوضح قاعدة تكوين الثوابت أو أسمائها وفقاً  
للأنماط المختلفة باستخدام خرائط القواعد .







شكل (٢ - ١)

## (٢ - ٣) الإعلان (Declaration) :

عرفنا أن أسماء البيانات (identifiers) تستخدم لتسمية الأوعية التي تحتوي على البيانات . والواقع أن هناك نوعين من البيانات أحدهما ثابت طوال البرنامج والآخر يتغير أثناء تنفيذ البرنامج .

فعلى سبيل المثال إذا كنا بصدد الحديث عن مساحة الدائرة فإن النسبة التقريبية  $\pi$  (pi) تكون دائماً 3.14159 (تقريباً) .

أما القيم الأخرى مثل مساحة الدائرة نفسها فهي تتغير حسب العملية الجارية بالبرنامج .

ويمكن الإعلان في البرنامج عن نوعية أسماء البيانات المزمع استخدامها لتكون إما ثابتة (CONST) أو متغيرة (VAR). ويطلق على أسماء البيانات الثابتة اسم الثوابت المسماة (named-constants). وهى نوعية من أسماء البيانات لا تتوفر بكل اللغات، ومن درس لغة فورتران يجد ما يناظرها تحت اسم البارامترات (parameters).

## (٢-٣-١) إعلان الثوابت المُسمَّاه (أسماء البيانات الثابتة) CONST

يمكن الإعلان عن أسماء البيانات الثابتة أو الثوابت المسماة (named-constants) في البرنامج كالآتي :

**CONST Pi = 3.141159;**

بهذا التعبير يمكن أن يفهم المترجم أن اسم البيان **Pi** سوف يظل ثابتاً طوال البرنامج وأنه من النوع الحقيقي لاحتوائه على علامة عشرية. فإذا ذكر الاسم **Pi** سوف يظل ثابتاً طوال البرنامج وأنه من النوع الحقيقي لاحتوائه على علامة عشرية. فإذا ذكر الاسم **Pi** فيما بعد أثناء البرنامج فإن المترجم يقوم باستبداله تلقائياً بالعدد 3.141159 ويجوز الإعلان عن ثابت مسمى يحتوي على ثابت حرفي ولكن يلزم استخدام علامات الاقتباس في هذه الحالة مثل :

**CONST line = '\*\*\*\*\*';**

## (٢-٣-٢) إعلان المتغيرات (أسماء البيانات المتغيرة) VAR

أما الإعلان عن المتغيرات فيتم باستخدام الكلمة **VAR** كالمثال الآتي :

**VAR temp :real;**

الاسم                  النمط

ويتكون الإعلان عن المتغيرات من الكلمة **VAR** متبوعة باسم المتغير ثم العلامة (:). يليها نمط المتغير المعلن عنه. والمتغيرات تنتمي إلى الأنماط القياسية الأربعة السابقة علاوة على نمط مصفوفة الحرفيات المُحمَّزة (packed array of

char أو نمط الحرفيات (string) وهذا الأخير قد لا تتوفر بكل الطرازات .  
والإعلان عن المتغير لا يتضمن منحه أية قيمة فعلية فهو حتى هذه النقطة  
فارغ بلا محتويات . ويجوز الإعلان عن عدة متغيرات بعبارة واحدة مع فصل  
المتغيرات عن بعضها البعض بفاصلة .

مثال (٢ - ١) :

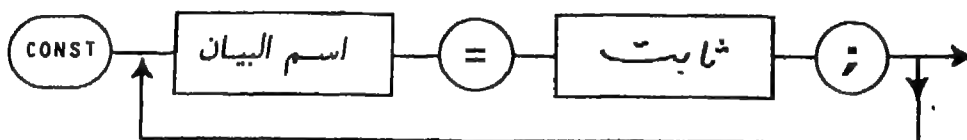
**CONST Pi = 3.14159;**

**VAR raduis, area : real;**

ومما يجدر بالملاحظة حتى الآن هو استخدام العلامات الخاصة ( ) ، ( = ) .

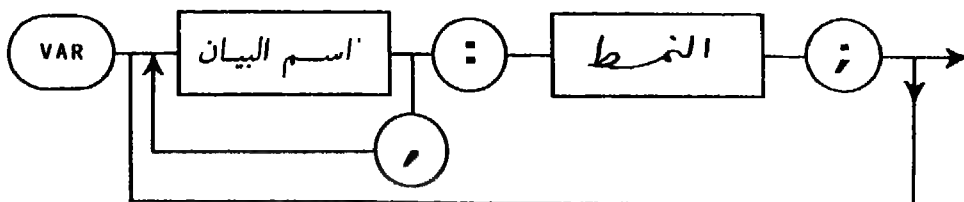
- فالعلامة (:) تستخدم للإعلان عن نمط اسم بيان .
- والعلامة (=) تستخدم لتحديد قيمة ثابت مسمى .

ويوضح الشكلان (٢ - ٢) ، (٢ - ٣) قاعدتي الإعلان عن المتغيرات :  
(VAR) والثوابت المسماه (const) بالخرائط .



شكل (٢ - ٢)

الإعلان عن أسماء البيانات الثابتة (أو الثوابت المسماه)



شكل (٢ - ٣)  
الإعلان عن أسماء البيانات المتغيرة (المتغيرات)

مثال (٢ - ٢) :

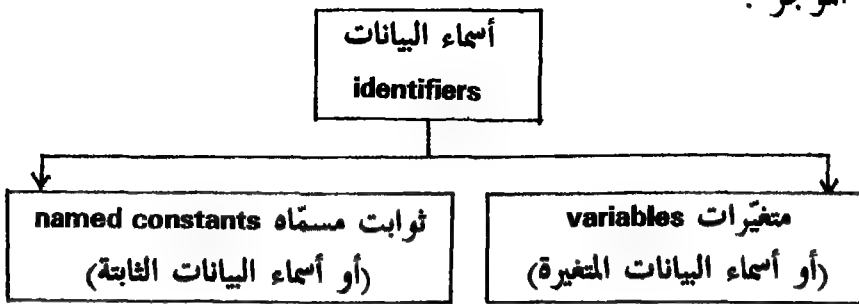
هذه شريحة من برنامج تستخدم للإعلان عن بعض المتغيرات VAR والثوابت

المسماه CONST :

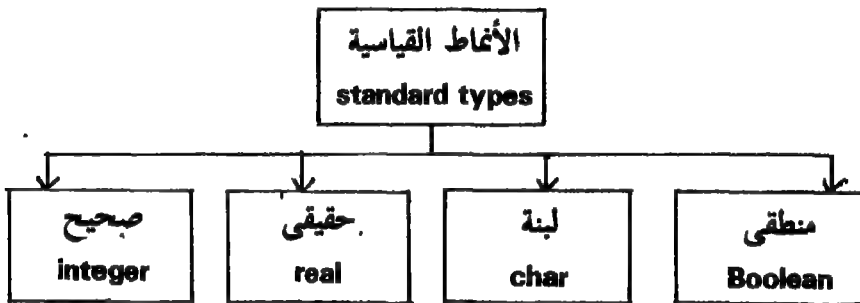
CONST	{	pi = 3.14159; (* type real *) zero = 0; (* type integer *) vrai = true; (* type boolean *) space = ' '; (* type char *) heading = 'Pascal program';	}	تعليقات
ثوابت مسماه				
VAR	{	radius : real; count : integer; firsttime : boolean; letter : char;	}	
متغيرات				

شكل (٢ - ٤)

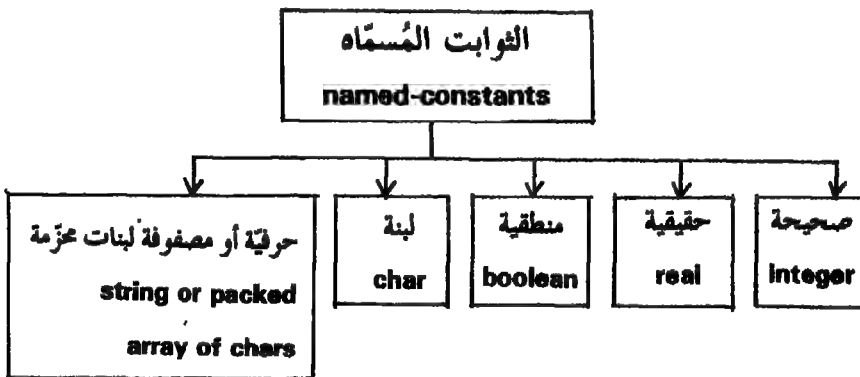
الموجز :



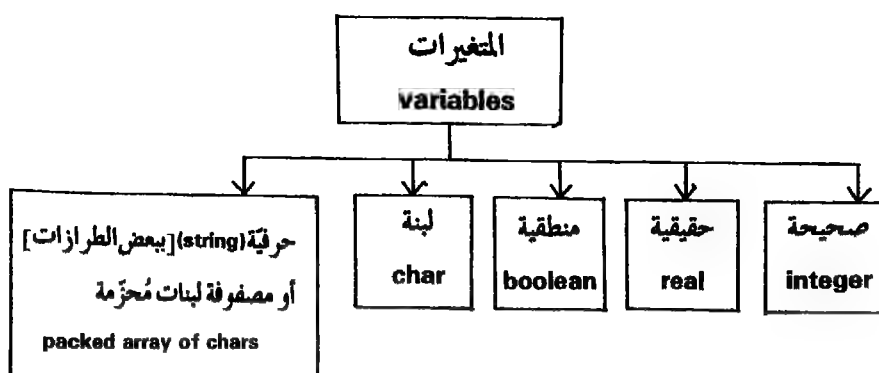
شكل (٢ - ٥)



شكل (٢ - ٦)



شكل (٢ - ٧)



شكل (٢ - ٨)

(٢ - ٤) التخصيص (Assignment) :

تعتبر المتغيرات أوعية فارغة حتى يتم « تعبئتها » بالبيانات . وأحد وسائل « التعبئة » هي عملية التخصيص التي يمكن التعبير عنها كما في المثال التالي شكل (٢ - ٩) :

مثال (٢ - ٣) :

الإعلان { CONST fixed = 5;  
VAR m, n : integer;

التخصيص { m := 3; "ثابت عددي"  
n := fixed; "ثابت مسمى"

علامة التخصيص

شكل (٢ - ٩)

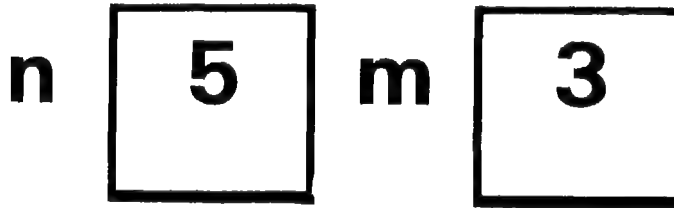
في هذه الشريحة من برنامج باسكال يعبر السطر الأول عن الإعلان عن ثابت مسمى (CONST) أطلقنا عليه الاسم fixed وحددنا له القيمة 5 .

أما في السطر الثاني فقد تم الإعلان عن متغيرين (VAR) هما  $m$  ,  $n$  وكلاهما من النوع الصحيح (أو النمط integer) .

أما السطرين الثالث والرابع فقد استخدمناهما للتخصيص للمتغيرات  $m$  ,  $n$  وقد ظهرت بهما علامة جديدة هي علامة التخصيص (=) .

في السطر الثالث قد خصصنا العدد 3 للمتغير  $m$  وفي السطر الرابع خصصنا الثابت المسمى fixed للمتغير  $n$  وحيث أن قيمة الثابت المسمى هي 5 فإن القيمة المختزنة في المتغير  $n$  قد أصبحت 5 .

ولو مثلنا المتغيرين  $m$  ،  $n$  بوعائين في الذاكرة فإن نتيجة عملية التخصيص السابقة سوف تؤدي إلى اختزان العدد 3 في الوعاء  $m$  والعدد 5 في الوعاء  $n$  .  
شكل (٢ - ١٠) .



شكل (٢ - ١٠)

وللتأكد من النتيجة السابقة يمكن استكمال شكل البرنامج وطباعة محتويات المتغيرين  $m$  ,  $n$  كما في شكل (٢ - ١١) .

كما يجوز نقل محتويات أحد المتغيرات إلى متغير آخر بواسطة عملية التخصيص أيضاً ، وتسمى هذه العملية بتخصيص متغير لمتغير . ونحن في غنى

عن القول بأن المتغير الذى يتم تخصيصه (الطرف الأيمن) يجب أن يكون معلوم القيمة أى سبق التخصيص له .

وفى شكل (٢ - ١٢) أضفنا علامة تخصيص جديدة هى :

**m := n;**

معنى ذلك أننا نقلنا محتويات الوعاء n (وهى 5) إلى الوعاء m وبذلك نرى أنه عند طبع محتويات المتغيرات بالبرنامج قد أصبح كل من m ، n يحتوى القيمة 5 .

ويجوز أيضاً تخصيص تعبير (expression) صحيح أو حقيقى أو منطقى لمتغير ، ولكن مهلاً فالتعبيرات موضوع مستقل سوف نؤفيه بحثاً فى الفقرات التالية .

CONST fixed = 5;  
VAR m, n : integer;

البرنامج

BEGIN

m := 3;  
n := fixed;

writeln('m=', m, ' n=', n)

أمر الطباعة

END.

Running  
m=3 n=5

التنفيذ

>

شكل (٢ - ١١)



```
CONST fixed = 5;
VAR m, n : integer;
```

البرنامج

```
BEGIN
```

```
m := 3;
n := fixed;
m := n;
```

تخصيص متغير لمتغير

```
writeln('m=', m, ' n=', n)
```

```
END.
```

```
Running
m=5 n=5
```

التنفيذ

شكل (٢ - ١٢)

## (٢ - ٥) التعبيرات والمؤثرات : Expressions & operators

التعبير هو عبارة عن علاقة مركبة بين الثوابت أو المتغيرات أو الدوال أو مجموعة منهم ويتم تكوين هذه العلاقة باستخدام المؤثرات .

فعند جمع ثابتين مثل 2,5 نكتب هذه العلاقة في صورة تعبير كالآتي :

$$2+5$$

وبدلاً من جمع الثوابت (الأعداد) يجوز أن نجمع المتغيرات مثل :

$$a+b$$

أو نجمع ثابتاً ومتغيراً معاً مثل :

$$a+5$$

كل هذه التعبيرات تسمى تعبيرات حسابية لأن المؤثر الذى ساهم فى بناء التعبير هو المؤثر الحسابى (+) .

وعلى ذلك نتوقع أن نوع المؤثرات المستخدمة هو الذى يحدد نوع التعبير الناتج .

ولنبداً بالتعبيرات الحسابية .

(٢-٥-١) التعبيرات الحسابية (الصحيحة والحقيقية) :

لكى نبني تعبيراً حسابياً — سواء كان من النوع الصحيح أو الحقيقى — فإننا نستخدم المؤثرات الحسابية التى نوجزها فيما يلى :

المؤثرات الحسابية	التأثير
+	للجمع (وعلامه الموجب)
-	للطرح (وعلامه السالب)
*	للضرب
/	لقسمة الحقيقية
	(مثال :
	ناتج القسمة $\frac{9}{4}$ هو 2.25)

وبالطبع فإن هذه المؤثرات تستخدم وفقاً لأولويات معينة فى التنفيذ كالآتى :

(١) الأقواس أولاً .

(٢) عمليات القسمة والضرب (أيهما يأتي أولاً من اليسار إلى اليمين)  
(ملاحظة : المؤثران DIM ، MOD ، لهما نفس الأولوية — سيأتي شرحهما) .

(٣) عمليتا الجمع والطرح (أيهما يأتي أولاً من اليسار إلى اليمين) .

مثال (١) : إذا أعطيت التعبير :

$$\frac{a + b}{c + d}$$

فإذا يمكن صياغته بلغة باسكال بالصورة الآتية :

$$(A + B)/(C + D)$$

فإذا أهملنا الأقواس أى كتبناه بالصورة :

$$A + B / C + D$$

فإن هذا يكافئ التعبير الرياضى :

$$a + \frac{b}{c} + d$$

وهو يختلف تماماً عن الصيغة الأصلية .

وبصفة عامة يمكن القول بأن استخدام الأقواس يجنبك الخطأ أى أن استخدام الأقواس من باب الاحتياط لا يضر بينما يتسبب نقص الأقواس فى أخطاء رياضية جسيمة .

مثال (٢) :

● ونذكر أيضاً المثال الآتى الذى يحتوى على أعداد يمكن حسابها :

$$4 + 3 * 2 - 6 / 2 = 7$$

$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$   
 ②   ①   ④   ③

← أولوية التنفيذ

● والمثال الآتي تستخدم فيه الأقواس فتأخذ الأولوية الأولى :

$$(4 + 3) * 2 - 6 / 2 = 11$$

$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$   
 ①   ②   ④   ③

← أولوية التنفيذ

● أنظر أيضاً هذا المثال عندما تتغير أماكن الأقواس :

$$(4 + 3 * 2 - 6) / 2 = 2$$

$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$   
 ②   ①   ③   ④

← أولوية التنفيذ

● وإذا استخدمنا عدة أقواس متداخلة فإن الأقواس الداخلية تفك أولاً :

$$( (4 + 3) * 2 - 6 ) / 2 = 4$$

$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$   
 ①   ②   ③   ④

← أولوية التنفيذ

● عند احتواء التعبير الرياضى على أقواس زائدة لا تتغير النتيجة كالآتى :

$$4 + (3 * 2) - (6 / 2) = 7$$

● المؤثرات الخاصة :

بخلاف المؤثرات السابقة فإن هناك المؤثرين DIV ، MOD اللذين يستخدمان مع الأعداد الصحيحة .

( ١ ) المؤثر DIV ويسمى مؤثر القسمة الصحيحة لأن ناتج القسمة يحذف منه الجزء الكسرى .

مثال : ناتج القسمة 4 DIV 19 هو 4

( ٢ ) مؤثر الباقي MOD : ويعطى هذا المؤثر باقى قسمة عدد صحيح على عدد صحيح (وهو اختصار كلمة modulo) .

مثال : التعبير 4 MOD 19 يعطى القيمة 3

أى أن القانون العام لهذا المؤثر هو :

$$a \text{ MOD } b = a - (a \text{ DIV } b) * b$$

وتأثير هذا المؤثر على القيم السالبة يعتمد على طراز اللغة .

والمؤثران DIV ، MOD لهما نفس الأولوية كمؤثرات القسمة الحقيقية والضرب .

ملاحظة :

حيث أن المؤثرات DIV ، MOD تستخدم مع الأعداد الصحيحة فقط لذلك يمكن تغيير النمط من « صحيح » إلى « حقيقى » متى دعت الحاجة إلى ذلك وذلك بتخصيص محتويات المتغير الصحيح لآخر حقيقى كالمثال الآتى شكل (٢ - ١٣) .

مثال (٢ - ٤) :

```
VAR a:real;
    b:integer;
```

البرنامج

```
BEGIN
```

```
  b:=3;
```

```
  a:=b;
```

```
  writeln('a=',a,' b=',b)
```

```
END.
```

تخصيص متغير صحيح لأخر حقيقي

```
>
```

```
Running
```

عدد حقيقي

عدد صحيح

التنفيذ

```
a= 3.000000000000E+00 b=3
```

شكل (٢ - ١٣)

ولكن لا يجوز تغيير النمط من « حقيقي » إلى « صحيح » ، حيث أن ذلك يتسبب في فقد البيانات . وإذا دعت الحاجة إلى ذلك يتم استخدام « دوال التحويل » ، التي نعرضها في الفقرة التالية .

مثال (٢ - ٥) حساب مساحة الحجرة :

وهذا المثال يعالج مرة أخرى « مساحة الحجرة » التي عرضناها في الباب الأول ولكن مع الاستفادة من المؤثرات الحسابية التي عرضناها بإدخال متغيرات جديدة هي :

المحيط (ضعف مجموع الطول والعرض)  
مساحة الأرضية (الطول في العرض)

perimeter  
floorarea

هذا علاوة على المتغيرات الأصلية وهي :

الطول length ، العرض width ، الارتفاع height .

ويتم إدخال قيمة هذه المتغيرات أثناء تشغيل البرنامج حيث تتم قراءتها بواسطة العبارة **read** وتخزينها في الذاكرة ( سيأتي شرحها ) .

والبرنامج يحسب ويطلع كلاً من المحيط ، مساحة الجدران ، مساحة الأرضية ، حجم الحجارة .

```

program roomsize3(input,output);

var length, width, height,
    perimeter, floorarea : integer;

begin
    writeln('type in length, width and height');
    read(length, width, height);
    perimeter := 2*(length + width);
    floorarea := length*width;
    {
        writeln('perimeter is ', perimeter);
        writeln('wall area is ', perimeter*height);
        writeln('floorarea is ', floorarea);
        writeln('room volume is ', floorarea*height)
    }
end.
    
```

*مساحة الحجارة*

*إدخال قيم المتغيرات الثلاثة*

*الحسابات والتقييمات*

*الطباعة مضمنة بعرض الحسابات*

شكل ( ٢ - ١٤ )

( ٢ - ٥ - ٢ ) الدوال القياسية (Standard functions) :

الدوال بصفة عامة هي برامج جاهزة توجد ضمن مترجم اللغة وتؤدي وظائف معينة يكثر الحاجة إليها في التطبيقات المختلفة ولكل دالة اسم خاص تستدعى به عند الحاجة إليها في البرنامج .

والدالة تتكون من اسم الدالة ، ودليل الدالة (أو البارامتر) كالمثال الآتي :

الدليل  $\text{trunc}(a)$  الاسم

فهذه الدالة اسمها trunc وهو اختصار كلمة (truncation) بمعنى « قطع » ويطلق عليها أيضاً دالة القطع أو دالة قطع الكسور ، وهي تستخدم لحذف الكسور من الأعداد الحقيقية .

أما الحرف a فهو دليل الدالة (argument) وهو يمثل العدد الذي تؤثر عليه الدالة . ولنر معاً المثال الآتي حيث نقطع العدد 5.566 بالدالة trunc .

مثال (٢ - ٦) :

VAR a:real:

begin  
a:=5.566;  
writeln trunc(a);  
END.

البرنامج

Running

5

العدد المقطوع

التنفيذ

شكل (٢ - ١٥)

والقطع معناه حذف الكسر تماماً وليس التقريب . أما التقريب فنستخدم له الدالة round كالمثال الآتي :

مثال (٢ - ٧) :



```
VAR a:real;
```

```
begin
  a:=5.566;
  writeln(round(a));
END.
```

البرنامج

Running

← العدد المقرب (التنفيذ)

شكل (٢ - ١٦)

وكما نرى فإن العدد 5.566 عندما أثرت عليه دالة التقريب فإنه قد تم تقريبه إلى أقرب رقم صحيح فأصبح 6 .

والشروط التي يجب توفرها في الدليل لهاتين الدالتين أن يكون من النوع الحقيقي وقد يكون ثابتاً أو متغيراً أو تعبيراً .

أما الناتج فهو عدد صحيح ويجوز تخصيصه لمتغير صحيح كالآتي :

```
b:=trunc(a);
```

```
b:=round(a);
```

حيث : a متغير أو تعبير حقيقي .

b متغير صحيح .

وقد يطلق اسم دوال التحويل (conversion functions) على كل من الدالتين trunc ، round لأنهما تستخدمان في التحويل من حقيقي إلى صحيح .

● كما تم لغة باسكال بالدوال القياسية الآتية :

الدالة	الوظيفة
<b>sqr(i)</b>	تعطى مربع الدليل i سواء كان حقيقياً أو صحيحاً
<b>abs(i)</b>	تعطى القيمة المطلقة للدليل i سواء كان حقيقياً أو صحيحاً

أمثلة :

<b>sqr(3)</b>	تعطى	9
<b>sqr(2.5)</b>	تعطى	6.25
<b>abs(- 25)</b>	تعطى	25
<b>abs(2)</b>	تعطى	2
<b>abs(- 27.12)</b>	تعطى	27.12

● أما المجموعة الآتية من الدوال فهي تعطى القيمة الحقيقية وتؤثر على أدلة حقيقية فقط :

الدالة	الوظيفة
<b>sqr(r)</b>	دالة الجذر التربيعى للدليل r
<b>sin(r)</b>	دالة جيب الزاوية r (r بالتقدير الدائرى) .
<b>cos(r)</b>	دالة جيب تمام الزاوية r (r بالتقدير الدائرى)
<b>arctan(x)</b>	الزاوية التى ظلها x (الجواب بالتقدير الدائرى) (فى طراز اللغة UCSD تسمى هذه الدالة atan)
<b>exp(x)</b>	الدالة الأسية للدليل x ( $e^x$ )
<b>ln(x)</b>	اللوغاريتم الطبيعى للدليل x

## (٢-٥-٣) المؤثرات الأسية (exponential operators) :

لا يوجد في لغة باسكال المؤثر الذي يرفع إلى أس كما في لغتي بيسك (١) وفورتران (\*\*). حيث أن هذا المؤثر يحتل زمناً ملموساً من وقت الكمبيوتر. ومع ذلك يمكن الاستعاضة عنه بدالة التربيع `sqr`.

والبرنامج التالي يوضح كيف يمكن استخدام هذه الدالة للحصول على المتغير `x` مرفوعاً لأسس مختلفة 2، 3، 4. وتظهر لنا عبارة القراءة `read` في السطر الأول بعد بداية البرنامج والتي تستخدم لمنح المتغير `x` قيمة عددية يستقبلها الكمبيوتر أثناء التنفيذ (وسوف نتحدث عنها بالتفصيل في حينه).

مثال (٢-٨) :

البرنامج :

```

program power(input,output);
  var x:integer;
begin
  read(x);
  writeln;
  writeln('if x= ',x,' then: ');
  writeln(' x^2= ',sqr(x));
  writeln(' x^3= ',x*sqr(x));
  writeln(' x^4= ',sqr(sqr(x)));
end.
```

استقبل قيمة المتغير `x`

التنفيذ :

```

2
if x= 2 then:
  x^2= 4
  x^3= 8
  x^4= 16
```

القيمة المدخلة للمتغير `x`

النتائج

شكل (٢-١٧)

ولعلنا نلاحظ أن دليل الدالة  $\text{sqr}$  يمكن أن يكون عبارة عن دالة أخرى وهذا ينطبق على سائر الدوال .

وبالإضافة إلى هذه الدالة فتوجد أيضاً دوال أخرى تفيد في الرفع إلى أس مثل الدالة اللوغاريتمية  $\ln$  والدالة الأسية  $\exp$  .

ويمكن إيجاد العلاقة  $a^b$  (a مرفوعة إلى الأس b) باستخدام التعبير :

$$\exp(\ln(a)*b)$$

مثال (٢ - ٩) سوبر ماركت :

مطلوب من عامل الخزينة « الكيس » ، في محلات « السوبر ماركت » الكبيرة ، أن يكون سريعاً في إجراء العمليات الحسابية ولكن مهما كانت سرعته فإن الكمبيوتر لا زال يتفوق بسرعته الهائلة في هذه العمليات . والبرنامج التالي يمكن استخدامه لمساعدة عامل الخزينة حيث يخبره بالمبلغ المتبقى من الجنيه في صورة عدد من الوحدات النقدية مثل عدد العشرة قروش والخمسة قروش والقروش إلى آخره .

ومدخلات البرنامج هي : سعر البضاعة المشتراه (price) .

```

program change(input,output);

var  price, change, noof50s, noof10s,
     noof5s, noof2s, noof1s : integer;

begin
  read(price);

  change := 100 - price;
  noof50s := change div 50;
  change := change mod 50;

  noof10s := change div 10;
  change := change mod 10;

  noof5s := change div 5;
  change := change mod 5;

  noof2s := change div 2;
  change := change mod 2;

  noof1s := change;

  writeln('change due is:  no of 50s      ', noof50s);
  writeln('                  no of 10s     ', noof10s);
  writeln('                  no of  5s      ', noof5s);
  writeln('                  no of  2s      ', noof2s);
  writeln('                  no of  1s      ', noof1s);

end.

```

سعر البضاعة  
عدد قطع الخمسين قرشاً  
الباقى  
سوبر ماركت  
الباقى بالعملة المختلفة

شكل (٢ - ١٨)

مثال (٢ - ١٠) التحويل من مئوى إلى فهرنهايت :

هذا البرنامج يقوم بقراءة درجة الحرارة بالتدرج وبحسب ما ينظرها بالتدرج  
الفهرنهايت وفقاً للعلاقة :

$$fdegrees = cdegrees \times \frac{9}{5} + 32$$

حيث fdegrees هى درجة الحرارة بالفهرنهايت .

cdegrees هى درجة الحرارة المئوية .

ثم يستخدم البرنامج الدالة round في طبع نتيجة تقريبية للعدد الناتج علاوة على القيمة الصحيحة فتكون إجابته كالمثال الآتي :

$$21.50c = 70.70f \text{ or approx } 71f$$

تقريباً

```
program conversion(input,output);
var cdegrees, fdegrees : real;
begin
    read(cdegrees);
    fdegrees := cdegrees * 9/5 + 32;
    writeln(cdegrees, 'c = ', fdegrees, 'f or approx ',
            round(fdegrees), 'f')
end.
```

التحويل من مئوية إلى فهرنهايت

شكل (٢ - ١٩)

## (٢ - ٦) التعبيرات المنطقية (Boolean Expressions) :

يعتمد التعبير المنطقي في تكوينه على المؤثرات «العلاقية» و « المنطقية » التي سيأتى شرحها وعادة تكون قيمة التعبير المنطقي أحد شيئين : صحيح (True) أو غير صحيح (false) .

وعادة يستخدم التعبير المنطقي للتعبير عن شرط ما قد يتحقق أو لا يتحقق .

## (٢ - ٦ - ١) المؤثرات العلاقية (Relational operators) :

تستخدم المؤثرات العلاقية في المقارنات بين القيم المختلفة للمتغيرات والتعبيرات وهي تشمل الآتي :

المؤثر	المعنى
$>$	أكبر من
$<$	أصغر من
$>=$	أكبر من أو يساوى
$<=$	أصغر من أو يساوى
$=$	يساوى
$< >$	ليس أكبر من

وفى الأمثلة الآتية نناقش استخدام هذه المؤثرات فى التعبيرات المنطقية :  
التعبير الآتى يعتبر تعبير منطقياً :

$$(a + b) < (c + d)$$

وعند تقييم هذا التعبير قد يعطى القيمة « صحيح » (true) إذا كان المجموع  $(a + b)$  أقل من المجموع  $(c + d)$  ، وإلا فإنه يعطى القيمة « غير صحيح » (false) .

ونلاحظ ضرورة وجود الأقواس فى هذا التعبير .

ولنر استخدام المؤثرات العلاقية مع التعبيرات المنطقية فى هذه الشريحة من البرنامج (شكل ٢ - ٢٠) :

مثال (٢ - ١١) :

```

var    a,b,c,d:real;
      first,
      second,
      same:boolean;
begin
  first:=a>b;
  second:=c<d;
  same:=first=second;
  read(a,b,c,d);
  :
  :
end.

```

تعبير منطقي

### شكل (٢ - ٢٠)

ففي هذا المثال بعد أن يتم الإعلان عن المتغيرات المنطقية الثلاثة `first` ، `second` ، `same` ، نبدأ في الشروط التي تمنح هذه المتغيرات القيمة الصحيحة `true` والتي إذا لم تحقق فإنها تحتوى على القيمة غير الصحيحة `false` .

فالمتغير `first` تكون قيمته `true` إذا كانت  $a > b$  . والمتغير `second` تكون قيمته `true` إذا كانت  $c < d$  أما المتغير `same` فتكون قيمته صحيحة إذا كان المتغيران `first` ، `second` متساويين . ولنجرب الآن تخصيص بعض الأعداد لهذه المتغيرات ونرى النتيجة شكل (٢ - ٢١) :

مثال (٢ - ١٢) :



```

var    a,b,c,d:real;
      first,
      second,
      same:boolean;
begin
  first:=a>b;
  second:=c<d;
  same:=first=second;
  a:=77 ;b:=60; c:=6; d:=54;
  write<first,' ',second,' ',same>
end.

```

البرنامج :

الاعلان

الشروط

التعيين

الطباعة

التنفيذ :

TRUE TRUE TRUE

شكل (٢ - ٢١)

وكما نرى فالنتيجة التي طبعها البرنامج هي :

TRUE TRUE TRUE

وهي تمثل القيم المنطقية التي تحتوى عليها المتغيرات المنطقية الثلاثة . وبالتأمل في الأعداد التي خصصناها للمتغيرات a,b,c,d نجد أنها جميعاً تحقق الشروط التي تجعل المتغيرات المنطقية صحيحة (true) .

هناك وسيلة أقوى للاستمتاع بهذا البرنامج وفهمه بعمق أكثر .

ماذا لو استخدمنا العبارة read لكي نستقبل قيماً مختلفة للمتغيرات a,b,c,d عند كل تنفيذ للبرنامج ؟ إننا بذلك يمكن أن نشاهد كل الاحتمالات المختلفة للنتائج . أنظر البرنامج المعدل شكل (٢ - ٢٢) مصحوباً بالتنفيذ لقيم مختلفة للمتغيرات .

ولنلاحظ عند إدخال المتغيرات أنها تدخل وفقاً لنفس الترتيب الوارد في العبارة read من اليسار إلى اليمين .

ففي التنفيذ الأخير مثلاً نجد أن  $a=4$  ،  $b=3$  ،  $c=2$  ،  $d=1$  .

وكذلك عند طبع قيمة المتغيرات المنطقية فإنها تطبع حسب الترتيب الوارد في عبارة الطبع writeln من اليسار إلى اليمين أيضاً .

مثال (٢ - ١٣) :

```
var    a,b,c,d:real;
      first,
      second,
      same:boolean;
begin
    first:=a>b;
    second:=c<d;
    same:=first=second;
    read(a,b,c,d);
    writeln;
    write(first,' ',second,' ',same)
end.
```

البرنامج :

التنفيذ :

القيم المدخلة للمتغيرات  $a, b, c, d$  بالترتيب من اليسار  
 2 2 2 1             
 TRUE FALSE FALSE            النتائج

1 1 1 1  
 FALSE FALSE TRUE

1 2 3 4  
 FALSE FALSE TRUE

9 7 1 4  
 TRUE TRUE TRUE

4 3 2 1  
 TRUE TRUE TRUE

شكل (٢ - ٢٢)

## مثال (٢ — ١٤) مقارنة الكميات الحقيقية :

عند استخدام الكميات الحقيقية يجب أن يضع المبرمج في اعتباره أنها لا يمكن تمثيلها في الكمبيوتر بدقة متناهية فجميع القيم الحقيقية تخضع للتقريب .  
لذلك فلا معنى لأن نختبر قيمتين من النوع الحقيقي بغرض معرفة ما إذا كانتا متساويتين أم لا .

وبدلاً من ذلك يتم اختبار الفرق بين القيمتين بحيث يكون أصغر من قيمة صغيرة جداً محددة سلفاً (وتختلف هذه القيمة بالطبع بحسب نوع التطبيق) .  
ولنر شريحة البرنامج التالية التي تختبر الفرق بين الكميتين الحقيقيتين a,b بحيث يكون أصغر من قيمة السماح (tolerance) المحددة بالكسر (3-1E) :

```
CONST    tolerance = 1.0e-3;
VAR      a,b : real;
          same : boolean;

....
same := abs(a-b) < tolerance;
```

## شكل (٢ — ٢٣)

## (٢ — ٦ — ٢) المؤثرات المنطقية Boolean Operators :

تستخدم المؤثرات المنطقية (علاوة على المؤثرات العلاقية) لبناء التعبيرات المنطقية .

والمؤثرات المنطقية هي :

NOT مؤثر النفي المنطقي  
AND مؤثر الضرب المنطقي  
OR مؤثر الجمع المنطقي

ومن كان حديث التعامل مع مثل هذه المؤثرات يمكنه الرجوع للملحق (هـ) حيث نشرح معناها تفصيلاً .

أما من ناحية أولوية هذه المؤثرات فإن مؤثر النفي المنطقي NOT يتمتع بالأولوية الأولى يليه مؤثر الضرب AND ثم مؤثر الجمع OR .  
فعلى سبيل المثال يمكننا كتابة التعبير الآتي :

**NOT a AND b OR c AND NOT d**

حيث a ، b ، c ، d كلها متغيرات منطقية .

فما معنى هذا التعبير ؟ لعلنا نستطيع الاقتراب من معناه أكثر إذا استخدمنا الأقواس لتوضيح الأولويات كالتالي :

**((NOT a) AND b) OR (c AND (NOT d))**

بهذه الصورة يصبح التعبير أكثر وضوحاً حيث نرى أن الأقواس الداخلية — وهي تتمتع بالأولوية الأولى — قد احتضنت الجزء (NOT a) . فأصبح هذا الجزء ذا أولوية أولى . أما المؤثر AND بطرفيه اللذين يربطهما ببعض فقد وقع داخل القوس ذي الأولوية الثانية . أما المؤثر OR الذي يقع خارج الأقواس فسوف يأتي دوره كأولوية ثالثة .

ومن الجدير بالذكر أن استخدام الأقواس هنا لمجرد التوضيح ومن يستطيع أن يرى الأولويات بلا أقواس فليستخدم التعبير الأول .

فماذا لو أردنا أن يسرى مفعول المؤثر NOT على كل مكونات التعبير المنطقي ؟ لابد من استخدام الأقواس في هذه الحالة . ولنر مثلاً آخر :

**I:= (a > b) AND firsttime**

تأخذ هنا القيمة True إذا كان التعبير الذى في الطرف الأيمن مساوياً للقيمة true (أى أن كل من طرفيه يساوى القيمة true) .

فإذا أردنا أن تأخذ ا القيمة true عندما يكون التعبير في الطرف الأيمن مساوياً للقيمة false فإن هذا النفي يلزم معه استخدام المؤثر NOT بالصورة الآتية :

**I:= NOT ((a > b) AND firsttime)**

هل لاحظت أن التعبير في الطرف الأيمن قد وقع بالكامل داخل قوسين يسبقهما المؤثر NOT ؟ .

هذا ضرورى لصحة التعبير المقصود ، فمن الخطأ أن ننفي التعبير بلا أقواس كالآتى :

**I:= NOT (a > b) AND firsttime**

فالتعبير في الطرف الأيمن هنا له معنى مختلف فالمؤثر NOT في هذه الحالة يؤثر على العلاقة التى بين القوسين فقط . أى أن هذا التعبير يقول :

« تكون قيمة a هى true إذا تحقق معاً الشرطان الآتيان :

١ — أن تكون قيمة التعبير (a > b) هى false (بمعنى ألا تكون a أكبر من b) .

٢ — وأن تكون قيمة المتغير firsttime هى true . »

(٢-٦-٣) قواعد « دى مورجان » :

أما العالم « أغسطس دى مورجان » (Augustus de Morgan) فله بصمات واضحة في علم الجبر البولياني أو الجبر المنطقي حيث وضع قاعدتين هامتين شملت لهما لغة باسكال بالصورة الآتية :

١ — التعبير الآتي :

**NOT (a OR b)**

يُعتبر مكافئاً للتعبير :

**NOT a AND NOT b**

٢ — والتعبير الآتي :

**NOT (a AND b)**

مكافئ للتعبير :

**NOT a OR NOT b**

وإثبات هذه القواعد ليس مجال اهتمامنا في هذا الكتاب . وهي بصفة عامة لا تهم إلا المتخصصين والرياضيين .

(٢-٦-٤) الدالة odd :

تعتبر الدالة odd من الدوال القياسية وهي تنتج قيمة منطقية إذا أثرت على تعبير صحيح (integer expression) وصيغتها كالآتي :

**odd (integer expression)**

الدالة

الدليل

(تعبير صحيح)

والنتيجة تكون صحيحة true إذا كانت قيمة التعبير عدداً فردياً أما إذا كانت قيمة التعبير عدداً زوجياً تصبح النتيجة false .

فالدالة الآتية تعطى القيمة true :

**odd (1)**

والدالة الآتية تعطى القيمة false :

**odd (128)**

ولنر البرنامج شكل (٢ - ٢٤) مصحوباً بالتنفيذ حيث يطبع قيم الدالتين  
السابقتين :

مثال :

```
begin
writeln (odd(1));
writeln (odd(128));
end.
```

البرنامج :

التنفيذ :

TRUE  
FALSE

شكل (٢ - ٢٤)

(٢-٦-٥) مقارنة اللغات :

لعل من درس لغة بيسك من قبل ينتظر بلهفة أن يرى كيف تتعامل لغة



باسكال مع الحرفيات أو مع اللبئات وكيف تتم العمليات المختلفة على التعبيرات الحرفية . والواقع أن لغة باسكال لا تحتوى على مؤثرات حرفية أو مؤثرات على اللبئات (character operators) ومع ذلك فيمكن استخدام اللبئات في التعبيرات المنطقية كالمثال الآتى :

**VAR B : boolean;**

**ch1,ch2 : char;**

**b := ch1 = 'a',**

. فما معنى العبارة الأخيرة ؟

إنها تقول بأن المتغير البوليائى b يمكن أن يأخذ القيمة true إذا كان متغير اللبنة ch1 يحتوى على الحرف 'a' .

ويمكن أيضاً إضافة العبارة الآتية :

**b := ch1 = ch2**

بمعنى أن المتغير b يأخذ القيمة true إذا تساوت محتويات كل من متغيري اللبنة ch1 ، ch2 .

ومع ذلك فموضوع معالجة الحرفيات واللبئات سوف نتعرض له بالتفصيل فى البابين السادس والسابع .

## ■ تمارين على الباب الثانى :

س (٢ - ١) : اكتب إعلاناً بلغة باسكال للمتغيرات التى تصلح لاحتواء كل نوع من أنواع البيانات التالية :

( أ ) درجات امتحان (أعداد صحيحة)

(ب) متوسط درجات عدة امتحانات .

(ج) شفرة حرفية 'A' أو 'O' لتوضيح مستوى الامتحان .

( د ) نتيجة إذا ما كان الطالب قد نجح أو رسب .

ثم اكتب عبارات تخصيص بلغة باسكال تمنح بها القيم الآتية للمتغيرات التى أعلنت عنها فى الفقرة السابقة :

( أ ) 51

(ب) 47.5

(ج) المستوى A

( د ) ناجح

س (٢ - ٢) : اكتب قيمة التعبيرات الآتية :

$$16/5$$

$$16 \text{ div } 5$$

$$16 \text{ mod } 5$$

$$19/5$$

$$19 \text{ div } 5$$

$$19 \text{ mod } 5$$

$$8 \text{ div } 3 * 3$$

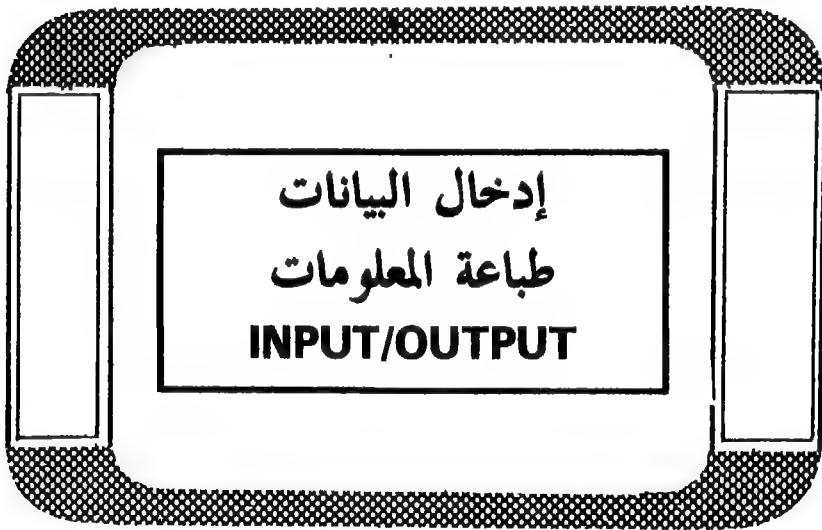
$$7 + 5 \text{ div } 3$$

$$13 - 5 \text{ mod } 3$$

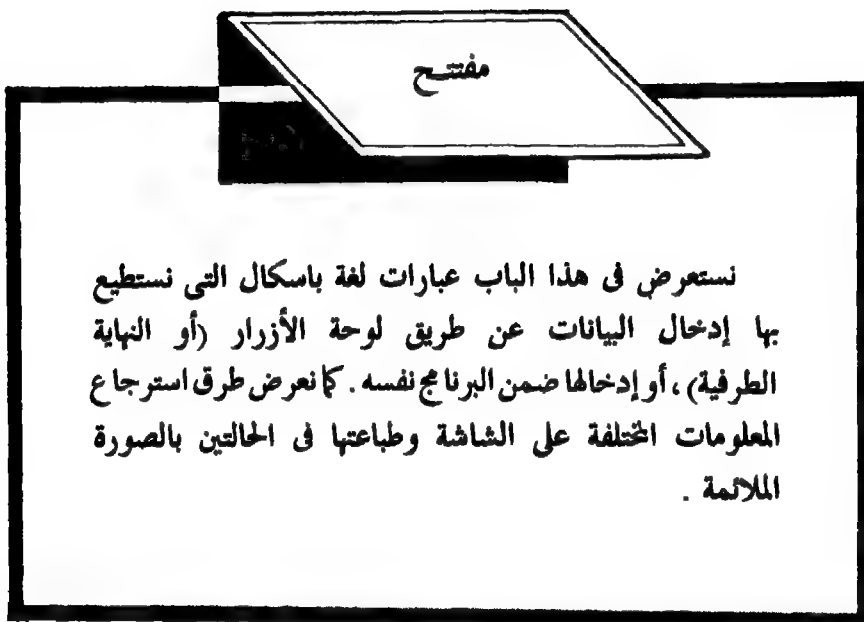


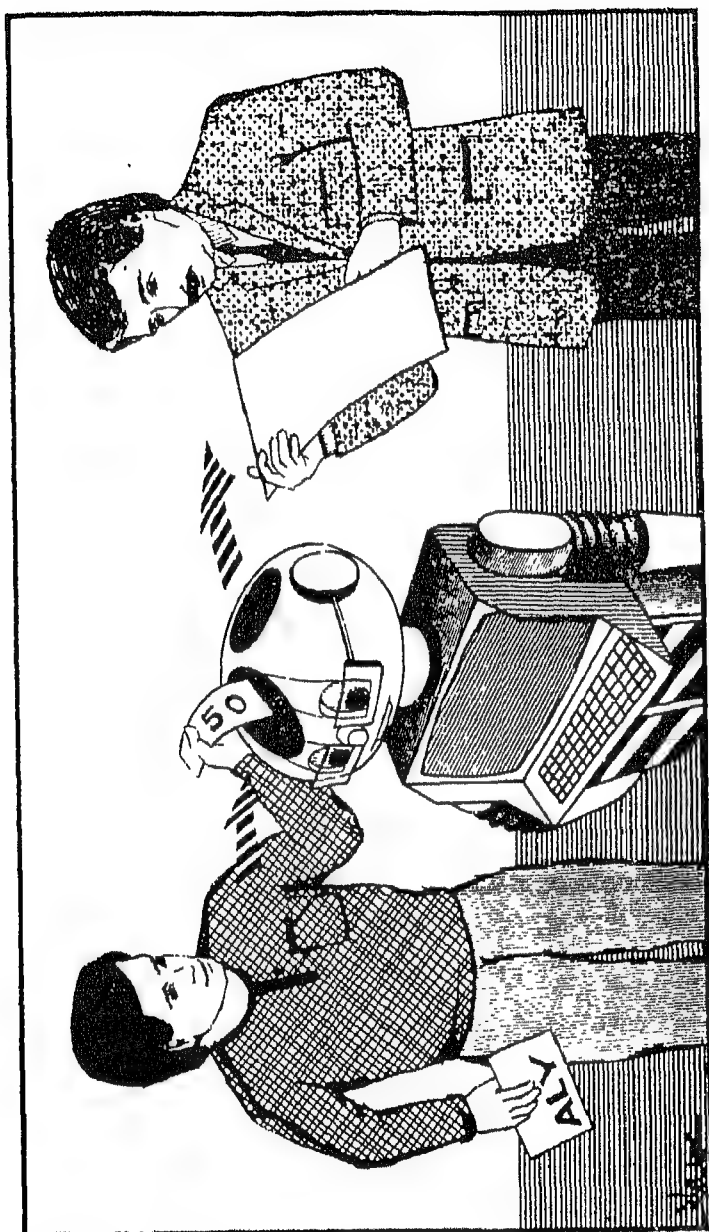


## الباب الثالث









إدخال البيانات واسترجاع المعلومات



### (٣ - ١) إدخال البيانات إلى البرنامج *read readln* :

نستطيع الآن أن نعود إلى البرنامج الذى عرضناه فى الباب الأول لنحدث عنه بمزيد من التفصيل .

مثال (٣ - ١) :

```
program roomsize(input,output);
var length, width, height : integer;
begin
    read(length, width, height);
    writeln('your room needs ', length*width,
            ' sq.m. of vinyl');
    writeln('and ', 2*(length + width)*height,
            ' sq.m. of wallcovering');
end.
```

مساحة الجبة

### شكل (٣ - ١)

فى هذا البرنامج أعلنّا عن ٣ متغيرات `length` , `height` , `width` كمتغيرات عددية صحيحة فى فقرة الإعلانات .

فإذا انتقلنا إلى العبارة الأولى فى الجزء التنفيذى من البرنامج والتى تلى الأمر `BEGIN` مباشرة سوف نلتقى بالعبارة `read` بمعنى (اقرأ) يليها أسماء المتغيرات الثلاثة بين قوسين .

عند تنفيذ هذه العبارة فإن الكمبيوتر يتوقف منتظراً من مستخدم البرنامج أن يدخل إليه ثلاثة أرقام عن طريق لوحة الأزرار وعندما يستقبل الكمبيوتر

الأرقام الثلاثة فإنه يُخصص للمتغيرات الثلاثة height , wridth , length بحسب ترتيب دخولها وبحسب ترتيب أسماء المتغيرات في العبارة read .

فالكومبيوتر بالطبع لا يعرف ما معنى length ؟ أو width ولكنه يعرف فقط أنه سيستقبل ٣ قيم لمتغيرات ثلاثة يخزنها في أماكنها في الذاكرة كل في وعاء معيّن كما في الشكل وهو يستخدم أسماء البيانات كعناوين يستدل بها على أوعية البيانات .

فالرقم الأول الذي ندخله سوف يُخصّصه الكومبيوتر للمتغير lenght لأنه هو المتغير الأول في العبارة read والرقم الثاني سيكون هو width والثالث هو height .

3	2	3	
---	---	---	--

length

width

height

شكل (٣ - ٢)

وبمجرد أن تمتلئ أوعية المتغيرات الثلاثة بالبيانات يبدأ الكومبيوتر في إجراء عمليات المعالجة اللازمة وهي حساب قيمة التعبيرات الحسابية الواردة في عبارات الطبع وهي :

$\text{length} * \text{width}$

$2 * (\text{length} + \text{width}) * \text{height}$

ونلاحظ ظهور العبارة (input) في مستهل البرنامج دلالة على أن هناك عملية إدخال بيانات بهذا البرنامج وعلى الكمبيوتر أن يعد العدة لذلك .

وعندما ندخل بيانات عديدة للكمبيوتر يجب أن تكون مفصولة عن بعضها البعض بمسافة خالية (أو أكثر) حتى يستطيع التمييز بينها (وهذا لا ينطبق على اللبئات كما سنرى فعند تنفيذ هذا البرنامج سوف نلتقى بعلامة استفهام في أقصى يسار الشاشة وعلينا أن ندخل الأرقام المطلوبة كالمثال الآتي :

**? 3 2 3**

إذا كانت هذه هي الأرقام المدخلة فسوف يوافقنا البرنامج بالرد الآتي فوراً :

**your room needs 6 sq.m. of vinyl  
and 30 sq.m. of wallcovering**

كما يمكن استخدام العبارة **readln** بدلاً من العبارة **read** ، والفارق بينهما هو أن العبارة **readln** تسمح بالانتقال إلى سطر جديد بعد إدخال البيانات وهي تغني عن استخدام العبارة **writeln** للانتقال إلى سطر جديد .

وبالطبع يمكن تنفيذ البرنامج أكثر من مرة ببيانات جديدة في كل مرة ، كالمثال الآتي :

**? 5 3 3** ← البيانات المدخلة

**your room needs 15 sq.m. of vinyl**

**and 48 sq.m. of wallcovering** ← إجابة الكمبيوتر

ألا ينقص هذا البرنامج شيء ؟

إننا الآن نعرف أن البرنامج عندما يتوقف منتظراً إدخال البيانات على يمين علامة الاستفهام — أنه يريدنا أن ندخل له ثلاثة أرقام متتالية تعبر عن الطول والعرض والارتفاع ولكن لو مرّ على إنشاء البرنامج فترة قصيرة فلا شك أننا سننسى ترتيب البيانات وربما نوعياتها أيضاً . هذا فضلاً عن أن شخصاً آخر لن يستطيع استخدام البرنامج بدون أن يراجع سطور البرنامج نفسه ليعرف ترتيب دخول هذه البيانات . والبرنامج الجيد يجب أن يكون سهل الاستخدام بمعنى أن تكون تعليمات استخدامه مطبوعة على الشاشة بحيث لا يتطلب الأمر قراءة البرنامج نفسه .

فما هي الإضافة المنتظرة التي تجعل البرنامج أكثر وضوحاً عندما يتوقف منتظراً منا إدخال البيانات ؟

(٣ — ٢) طباعة رسالة عند إدخال البيانات :

ماذا لو ظهرت علامة الاستفهام مسبقة ببعض العبارات التوضيحية كالمثال الآتي :

الرسالة ← type in length, width, and height  
البيانات المدخلة ← 4 2 3 ?

إن العبارة (أو العبارات) التي سبقت علامة الاستفهام والتي نصطلح على تسميتها بالرسالة (message) تدل المستخدم على نوعية وترتيب البيانات التي ينتظرها الكمبيوتر ففي هذه الحالة سوف يفهم المستخدم أن الكمبيوتر ينتظر إدخال ثلاثة أعداد متتابعة تمثل الطول والعرض

والارتفاع بالترتيب .

كيف يمكن تحقيق ذلك .

يمكن إضافة عبارة لطباعة هذه الرسالة قبل عبارة الدخول (read) مباشرة ،  
هذه العبارة هي :

**writeln('type in length, width, and height');**

بذلك يكون البرنامج في صورته الكاملة كالآتي :

مثال (٣ - ٢) :

```
program roomsize2(input,output);
var length, width, height : integer;
begin
    writeln('type in length, width and height');
    read(length, width, height);

    writeln('your room needs ', length*width,
            ' sq.m. of vinyl');
    writeln('and ', 2*(length + width)*height,
            ' sq.m. of wall covering')
end.
```

مساحة الجدران

شكل (٣ - ٢)

### (٣ — ٣) طباعة الرسائل والنتائج **Writeln** :

رأينا في الفقرة السابقة أن العبارة **writeln** تستخدم في طباعة رسالة ما على الشاشة . وأن الرسالة المطلوب طباعتها توضع بين علامتي اقتباس كقاعدة عامة . فكل ما تضعه بين علامتي الاقتباس يظهر مطبوعاً على الشاشة بما يحتويه من فواصل ومسافات خالية .

وقد أشرنا من قبل أن العبارة **writeln** تعني طباعة ما يأتي بعدها (بين قوسين) على سطر مستقل .

كما التقينا من قبل بالعبارة **writeln** في الباب الأول وعلمنا أن من خصائصها أيضاً إجراء بعض العمليات الحسابية أو تقييم التعبيرات قبل طباعتها وهذا واضح في السطرين الأخيرين من البرنامج .  
فالسطر الأول يقول :

**writeln('your room needs', length \* width,  
'sq.m. of vinyl');**

نرى أن هذه العبارة تحتوى على رسالتين وضعت كل منهما بين علامتي اقتباس وهما :

**your room needs**  
**sq.m. of vinyl**

كما تحتوى العبارة على التعبير الحسابى الذى أتى بدون علامات اقتباس :

**length\*width**

وعندما يصادف الكمبيوتر ضمن محتويات الطبع جزءاً بدون علامتى اقتباس ، يتوقع أن يكون تعبيراً حسابياً يحتاج لمعالجة وطريقة مختلفة فى الطباعة ، ولتر هذه الأمثلة :

مثال (٣ - ٣) :

العبارة	النتيجة المطبوعة
<b>writeln(25)</b>	<b>25</b>
<b>writeln(3*2)</b>	<b>6</b>
<b>writeln('3 x 2 = ',3*2)</b>	<b>3 x 2 = 6</b>

شكل (٣ - ٣)

نخرج من المثال السابق بالملاحظات الآتية :

١ - إذا كان البيان المطلوب طبعه بياناً عددياً (مثل ٢٥) أو تعبيراً حسابياً (مثل 3\*2) فلا يوضع البيان بين علامتى اقتباس والتعبير الحسابى يتم تقييمه

قبل طباعته .

٢ — إذا كان البيان المطلوب طبعه رسالة أو بصفة عامة بياناً حرفياً (string) فإن البيان يوضع بين علامتى اقتباس .

٣ — يجوز أن تشتمل عبارة الطبع على بيانات مختلفة (عددية وحرفية) وتفصل البيانات عن بعضها البعض بفصلة .

مثال (٣ — ٤) :

النتيجة المطبوعة	العبارة
15	<code>writeln(length * width)</code>
area is 15	<code>writeln('area is ' length * width)</code>

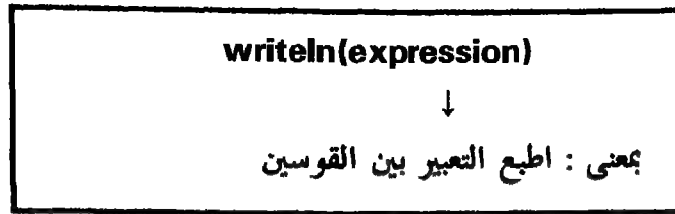
في هذا المثال أخذ التعبير الحسالى صورة مختلفة حيث احتوى على متغيرات بدلاً من الأعداد . ويقوم الكمبيوتر في هذه الحالة باسترجاع القيمة العددية لهذه المتغيرات من خانات الذاكرة قبل تقييم التعبير الحسالى .

ولقد أشرنا من قبل أن الأعداد المطبوعة في الخرج (output) ربما تحتوى على مسافات خالية قبلها أو بعدها بحسب طراز اللغة .

● صيغة عبارة الطباعة `Writeln` :

من الأمثلة السابقة نرى أن عبارة الطباعة تتبع الصيغة الآتية :





والتعبير الذى يقع بين قوسى عبارة الطباعة قد ينتمى إلى أحد الأنماط الآتية :

- ١ — حقيقى .
- ٢ — صحيح .
- ٣ — منطقى (فيما عدا بعض الطرازات) .
- ٤ — لبنة .
- ٥ — مصفوفة الحرفيات المحرمة (packed array of char) فإذا احتوى القوس على أكثر من تعبير فإن التعبيرات تفصل عن بعضها البعض بفاصلة كما رأينا فى الأمثلة السابقة .
- وإذا احتوت اللغة على المتغيرات الحرفية string فإنها يجوز طباعتها أيضاً .

### (٣ — ٤) عبارة الخرج : **Write** :

قد نقول « عبارة الطبع » أو « عبارة الخرج » وكلاهما يؤدي نفس المعنى فكلاهما تعنى استرجاع المعلومات من ذاكرة الكومبيوتر وإخراجها على أحد أجهزة الخرج مثل الشاشة أو جهاز الطباعة .

وعبارة الطبع الجديدة write تؤدي نفس الغرض لكن فارقاً واحداً يميزها عن عبارة الطبع writeln وهو أنها لا تجعل البرنامج ينتقل إلى السطر التالى . فلو تابعت عدة عبارات منها فإن محتويات العبارات جميعاً تطبع على نفس السطر .

مثال (٣ - ٥) :

العبارة	النتيجة المطبوعة
write ('my name is '); write ('Aly Hassan')	my name is Aly Hassan

شكل (٣ - ٥)

فكما نرى أن عبارتي الطبع اللتين أتينا متتاليتين قد نتج عنهما سطر واحد في خط الطبع .

وعلى ذلك نتوقع أن تؤدي العبارة :

**writeln;**

إلى ترك سطر خالي دون طباعة أي شيء . ومن خبر لغة بيسك من قبل لعله يلاحظ التشابه بين عبارة الطبع **PRINT** في لغة بيسك والعبارة **writeln** في لغة باسكال .

وينطبق على العبارة **write** ما ينطبق على العبارة **writeln** من قوانين .

مثال (٣ - ٦) :

النتيجة المطبوعة	العبارة
1st line 2nd line	<code>write ('1st line'); writeln; writeln('2nd line')</code>

شكل (٣ - ٦)

(٣ - ٥) استخدام عبارة الطبع في الرسم :

يمكن باستخدام عبارة الطبع `write` أو `writeln` رسم ما نشاء من الأشكال على الشاشة باستخدام اللبئات المختلفة التي تتيحها اللغة .

فبين علامتى الاقتباس يمكنك أن تضع ما تشاء من اللبئات مع قيد واحد على استخدام علامة الاقتباس نفسها حيث يتطلب الأمر استخدام علامتين متتابعتين كما عرضنا في مستهل الباب الأول .

والبرنامج التالى يرسم مثلنا باستخدام عبارة الطباعة :

مثال (٣ - ٧) :

```
program triangle(output);
```

البرنامج

```
begin
```

```
    writeln('          *');
    writeln('        *  *');
    writeln('      *    *');
    writeln('    *      *');
    writeln('  *        *');
    writeln(' *          * triangle');
    writeln(' *          *');
    writeln(' *          *');
    writeln('* * * * * * * * * * * * * * * *');
```

```
end.
```

شكل (٣ - ٧) أ

```
          *
        *  *
      *    *
    *      *
  *        *
 *          * triangle
 *          *
 *          *
 *          *
 * * * * * * * * * * * * * *
```

التنفيذ

شكل (٣ - ٧) ب

هل لاحظت في البرنامج السابق ما جاء في السطر الأول مع اسم البرنامج  
? triangle

لقد عقب اسم البرنامج العبارة (output) منفردة لأن هذا البرنامج لا يحتاج

لآية مدخلات (inputs) فهو مخصص للطباعة فقط !

مثال (٣ - ٨) طباعة فاتورة الكهرباء :

فلنفرض أن فاتورة الكهرباء قد جاءتك بهذا المنظر المبهم  
شكل (٣ - ٨) .

```
*****
present meter reading      6015
previous meter reading     5899
units used                  116
rate per unit               3.4p
standing charge             £1.14

the sum due is             £5.08
*****
```

شكل (٣ - ٨)

تري ما هو البرنامج الذى فى خلفية هذه الفاتورة ؟

هناك بعض البيانات يحتاجها البرنامج بالضرورة مثل القراءة الحالية  
(present meter reading) والقراءة السابقة (previous meter reading)  
وهناك بيانات تحتاج لإجراء بعض المعالجات الحسابية مثل المبلغ المطلوب  
(the sum due) والكهرباء المسحوبة (units used) وهناك بعض البيانات  
الثابتة التى تستخدم مع كل الفواتير مثل سعر الكهرباء (rate per unit)  
والمصروفات الثابتة (standing charge) .

لذلك فالبرنامج الذى نحن بصددده يحتاج لقراءة بعض البيانات من الخارج  
وهى القراءة السابقة والقراءة الحالية ثم يقوم بإجراء بعض الحسابات ثم يطبع  
الناتج بالصورة المطلوبة .

وهذا هو البرنامج :

```

program electricitybill(input,output);
var present, previous : integer;
begin
    read(present, previous);
    writeln('*****');
    writeln('present meter reading      ', present);
    writeln('previous meter reading      ', previous);
    writeln('units used                      ',
            present - previous);
    writeln('rate per unit                      3.4p');
    writeln('standing charge                    £1.14');
    writeln;
    write ('the sum due is              £');
    writeln( (present - previous)*3.4/100 + 1.14 );
    writeln('*****')
end.

```

البرنامج

فاتورة الكهرباء

### شكل (٣ - ٩)

والملاحظة التي نخرج بها من هذا البرنامج هي كيفية استخدام المسافات الخالية لتنسيق الأعمدة والخانات في الفاتورة النهائية المطلوب طباعتها .

### (٣ - ٦) صياغة الخرج : *Formatted output*

هذه هي طريقة بديلة للحصول على الخرج مطبوعاً وفقاً للصورة التي نراها ، وتسمى بالطباعة المصاغة أو الخرج المصاغ (formatted) .

ولنر العبارة الآتية :

**writeln(2\*256:8)**

الصيغة

إن الرقم 8: الذى جاء فى مؤخرة عبارة الطبع يخبر الكمبيوتر بأن يخصص الطباعة الناتج ثمانى خانات . وعلى ذلك يقوم الكمبيوتر بإجراء العملية الحسابية  $2*256$  حيث يكون الناتج هو 512 أى يحتل ثلاث خانات ، فيترك الكمبيوتر خمس مسافات خالية ثم يطبع العدد 512 . وهذه الطريقة تعتبر طريقة فعالة لتنظيم الطباعة لا سيما إذا كانت السطور المتتالية تنتظم فى أعمدة . وإذا أردنا كتابة البرنامج الذى استخدمناها من قبل فى رسم مثلث النجوم — بهذه الطريقة فإنه يصبح كالتالى . كما فى شكل (٣ — ١٠) :

مثال (٣ — ٩) :

**program triangle(output);**

**begin**

```
writeln('*':17);
writeln('*':15,'*':4);
writeln('*':13,'*':8);
writeln('*':11,'*':12);
writeln('*':9,'*':16);
writeln('*':7,'triangle':13,'*':7);
writeln('*':5,'*':24);
writeln('*':3,'*':28);
writeln('*****')
```

**end.**

شكل (٣ — ١٠)

## ● صياغة الطباعة مع الأعداد الحقيقية :

أما مع الأرقام الحقيقية فالصيغة تحتوى على جزئين الجزء الأول يمثل عدد خانات العدد كلها (عدد اللبئات المطلوب طبعا) أما الثانى فيمثل عدد الأرقام العشرية التى تقع على يمين العلامة .

والمثال التالى الموضح بالبرنامج شكل (٣ — ١١) يوضح كيفية صياغة الأعداد الحقيقية . فقد خصصنا عددين للمتغيرين  $x, y$  ثم طبعناهما وفقاً للصيغة الواردة فى عبارة الطبع `writeIn` . وللتأكد من النتيجة التى نحصل عليها فقد طبعنا سطرأ من النجوم فوق الأعداد حتى تظهر خانة كل رقم بوضوح .

فمثلاً العدد الأول **9.4790** يحتوى على ٦ خانات بما فيها خانة العلامة العشرية لكنه ظهر فى الخرج ممثلاً فى ٨ خانات أى مسبقاً بخانتين خاليتين (أنظر تنفيذ البرنامج) وذلك لأن الصيغة التى تم طبعه بها تحتوى على الرقم 8 يليه الرقم 4 . والرقم 8 يعنى عدد خانات الأرقام الصحيحة بما فيها العلامة ، والرقم 4 يعنى عدد الخانات العشرية كذلك نجد أن الفاصل بين العدد الأول والثانى نتج من أن العدد الثانى 3.14159 ظهر مطبوعاً فى عشر خانات وفقاً للصيغة المحددة له (5:10) ولذلك نجد أن هذا الفاصل يحتوى على ثلاث خانات أو ثلاث نجوم وهو الفرق بين عدد الأرقام الفعلية للعدد وبين رقم الصيغة (10:). .

مثال (٣ — ١٠) :



```
var x,y,z:real;
begin
x:=9.479;y:=3.141592;
writeln('*****');
writeln(x:8:4,y:10:5,x+y:9:2,
end.
```

البرنامج

```
*****
9.4790    3.14159    12.62
>
```

النتيجة

شكل (٣ - ١١)

مثال (٣ - ١١) :

الآن يمكننا تمثيل برنامج فاتورة الكهرباء باستخدام الطباعة المصاغة كالآتي :

```
writeln('present meter reading ', present:7);
writeln('previous meter reading', previous:7);
writeln('units used           ', present-previous:7);
writeln('rate per unit         ', 3.4 :6:1, 'p');
writeln('standing charge      £', 1.14:7:2);
write  ('the sum due is      £');
writeln( (present-previous)*3.4/100 + 1.14 :7:2 )
```

شكل (٣ - ١٢)

## ■ تمرينات على الباب الثالث :

س (٣ - ١) : في مثال فاتورة الكهرباء الذى عرضناه قد يكون من المناسب أن نستخدم الثوابت المسماة (named constants) بدلاً من الأرقام حتى يكون البرنامج أكثر عمومية ويسهل تغييره في المستقبل إذا طرأ أى تغيير على الأرقام اكتب البرنامج مرة أخرى باستخدام الثوابت المسماة والطباعة المصاغة .

س (٣ - ٢) اكتب برنامجاً يقرأ درجات تلميذ في أربعة امتحانات مختلفة (أرقاماً صحيحة) ويطبع متوسط الدرجات .

س (٣ - ٣) اكتب برنامجاً يقرأ عددين صحيحين ويطبع مجموعهما وحاصل ضربهما في صورة معادلتين كالمثال الآتي :  
إذا كان العددان هما :

$$4 \quad 7$$

فإن الخرج يكون على الصورة :

$$4 + 7 = 11$$

$$4 * 7 = 28$$

س (٣ - ٤) : اكتب برنامجاً يقرأ عدد ساعات العمل لموظف ما ، والأجر في الساعة ، وعدد الساعات الإضافية . علماً بأن أجر الوقت الإضافي ١,٥ مرة قدر معدل الأجر العادى ، ثم اطبع المرتب الإجمالى .

س (٣ - ٥) : اكتب برنامجاً يكتب الحرف الأول من اسمك باستخدام النجوم (\*).

س (٣ - ٦) : اكتب برنامجاً يستقبل رصيدك في البنك والمبلغ المسحوب  
ويطبع الرصيد النهائي بعد عملية السحب .

س (٣ - ٧) : اكتب برنامجاً يقرأ ثمن السلعة الإجمالي ونسبة الخصم (نسبة  
مئوية) ويقوم البرنامج بطبع فاتورة البيع كالمثال الآتي :

```
*****
gross price      $56.25
discount rate    2.5%
discount         $ 1.41
discount price   $54.84
*****
```

شكل (٣ - ١٣)

س (٣ - ٨) : وفقاً للنظرية النسبية فإن الأجسام المتحركة بسرعة كبيرة  
قريبة من سرعة الضوء ينكمش طولها في اتجاه سرعتها . وهذا يُعبر عنه بالمعادلة  
الآتية :

$$l = l_0 \sqrt{1 - \frac{v^2}{c^2}}$$

- حيث : ا هو طول الجسم أثناء الحركة  
 l0 هو طول الجسم ساكناً  
 v سرعة الجسم مقاسة بالمتري / ثانية .  
 c سرعة الضوء، وهي تساوى :

**299 792 458 متر / ثانية**

اكتب برنامجاً بلغة باسكال للإعلان عن المتغيرات المتضمنة بهذه العلاقة .  
 وبفرض معلومية طول الجسم ساكناً ، وسرعته اكتب عبارة (أو عبارات)  
 التخصيص اللازمة لحساب طول الجسم أثناء الحركة .



## الباب الرابع

### التفرع والتكرار **Branching & Looping**



## مفتتح

إن كل ما صادفناه حتى الآن من برامج عبارة عن بعض العمليات الحسابية والقرارات المنطقية البسيطة التي يستطيع الإنسان أداءها بدون احتياج حقيقى للكمبيوتر . لكننا لو اعتبرنا مشكلة واقعية مثل حساب الأجور لألف موظف يعملون فى شركة كبيرة لوجدناها مشكلة حقيقية تتطلب جهداً هائلاً ...

وربما كان حساب الأجر لموظف واحد أمر بسيط لا يستغرق دقائق ولكن عندما يزيد العدد إلى هذا الحد فسوف تواجهنا مشكلتان .. الأولى هى السرعة والثانية هى العمل المتكرر الذى يبعث على الملل وربما يسفر عن أخطاء تحتاج لمراجعات كثيرة .

وهذا هو دور الكمبيوتر الحقيقى .. فهو لا يمل ولا يتعب من التكرار ، ولا يخطئ طالما برمجته بالمنطق السليم ، هذا فضلاً عن سرعته المذهلة .

وفى هذا الباب نتحدث عن الوسائل المختلفة التى نستفيد بها من هذه الخصائص التى يتمتع بها الكمبيوتر .

*while for repeat*

#### (٤ - ١) عبارات التحكم *Control statements* :

رأينا في الأبواب السابقة أن بعض عبارات لغة باسكال تستخدم في إدخال البيانات والبعض الآخر في إخراج المعلومات وطباعتها ، كما خیرنا عبارات التخصیص والتعبیرات بأنواعها التي تعتبر الأداة الرئيسية في معالجة البيانات .

أما العبارات التي نحن بصدد عرضها الآن فهي تهدف إلى حسن استخدام إمكانيات الكمبيوتر بالاستفادة من سرعته في إجراء العمليات ، وهي تسمى عبارات التحكم . وأولى عمليات التحكم هي عملية التكرار .. وما أكثر ما يحتاج الكمبيوتر إلى التكرار فهذا هو أحد الأهداف الرئيسية من استخدامه .

ومن عبارات التحكم أيضاً تغيير مسار البرنامج وفقاً لتحقيق شرط ما . وقد نطلق على هذه العملية تفريع البرنامج .

#### (٤ - ٢) الحلقات التكرارية *For-statement* :

لو أردنا أن يطبع الكمبيوتر رسالة ما عدة مرات متتالية فهناك حل سهل بأن نكتب عبارة الطبع بعدد مرات التكرار المطلوبة . ولكن هذا الحل قد يؤدي بنا إلى كتابة برنامج طويل جداً بلا داع .

الطريقة البديلة هي استخدام العبارة *for* كما في المثال شكل (٤ - ١) حيث يقوم البرنامج بطباعة الرسالة خمس مرات متتالية .

مثال (٤ - ١) :



```

program demo(output):
  var i:integer;
begin
  for i:=1 to 5 do
  begin
    writeln('SORRY..I DON'T HEAR YOU ...SAY AGAIN');
    writeln
  end
end.

```

البرنامج :

متغير العداد

بداية الحلقة

نهاية الحلقة

التنفيذ :

```

SORRY..I DON'T HEAR YOU ...SAY AGAIN
SORRY..I DON'T HEAR YOU ...SAY AGAIN
SORRY..I DON'T HEAR YOU ...SAY AGAIN
SORRY..I DON'T HEAR YOU ...SAY AGAIN
SORRY..I DON'T HEAR YOU ...SAY AGAIN

```

شكل (٤ - ١)

كيف تعمل عبارة التكرار for ؟

نرى في السطر الأول (بعد بدء البرنامج بالعبارة begin) عبارة التكرار بالصورة الآتية :

**for i:= 1 to 5 do**

أما المتغير  $i$  الذى أعلنه فى البداية كمتغير صحيح فهو يعمل كعداد لعدد مرات التكرار . وقد بدأ هنا بالقيمة ( $i=1$ ) وسوف يوقف التكرار عندما تصل قراءته 5 .

وتشتمل العبارة for على كلمات ثلاث هى do ، to ، for ، وبلى العبارة for الحلقة التكرارية التى تحتوى على العملية المطلوب تكرارها وهى أمر الطباعة فى حالتنا هذه . ونلاحظ أن الحلقة التكرارية لها بداية begin ونهاية end بصرف النظر عن بداية ونهاية البرنامج الأسمى الذى تنتمى إليه .

وهناك عدة ملاحظات على قواعد اللغة فى هذا البرنامج :

( ١ ) لا تستخدم الفاصلة المنقوطة (semicolon) بعد كلمة do لأن عبارة for لا تكتمل إلا عندما تصل إلى نهايتها (end) . أما ما يقع بينهما فيعتبر جزءاً واحداً مستقلاً عن عبارات البرنامج .

ولذلك يمكن القول بأن البرنامج يتعامل مع الحلقة التكرارية التى تقع ما بين begin ، end كأنها عبارة واحدة أو بمعنى آخر فهذا البرنامج يمكن اعتباره محتوياً على عبارة واحدة لذلك لم نستخدم الفاصلة المنقوطة بعد العبارة end التى تمثل نهاية هذه العبارة .

ولو كان البرنامج يحتوى على عبارة أخرى بعد نهاية الحلقة لأصبح من الضروري استخدام الفاصلة بعد end لفصل العبارتين .

( ٢ ) ما بداخل الحلقة التكرارية يعتبر برنامجاً قائماً بذاته له بداية ونهاية وتتابع فيه العبارات مفصولة بالفاصلة المنقوطة . ويسرى على الحلقة ما يسرى على البرنامج من قواعد .

( ٣ ) مما يجدر بالملاحظة أن الإفراط فى استخدام الفاصلة المنقوطة لا يؤثر على سلامة البرنامج من الناحية اللغوية ولكن عدم وجود الفاصلة متى كان وجودها ضرورياً يمنع تنفيذ البرنامج تماماً .

(٤) ماذا يحدث لو حذفنا بداية (begin) ونهاية (end) الحلقة التكرارية ؟ .

إن فائدة البداية والنهاية كفاءة الأقواس في التعبيرات الحسابية أى أنها تحدد العبارات المقصودة بالتكرار . فإذا حذفنا البداية والنهاية اقتصر التكرار على أول عبارة تالية للكلمة do .

(٥) يبدأ العدّاد من قيمة ابتدائية (1) وينتهي عند قيمة نهائية (5) وبتغيير القيمة النهائية يمكن تغيير عدد مرات التكرار وهذا لا يتطلب سوى إصلاح بسيط في البرنامج .

بل إن هذه القيمة النهائية يمكن أن تكون متغيراً يقوم البرنامج بقراءته بالعبارة read وبذلك نستغنى تماماً عن إصلاح البرنامج وإعادة ترجمته .

مثال (٤ - ٢) :

```
var f,i:integer;
begin
  read (f);
  for i:=1 to f do
    begin
      writeln('sorry...say again');
      writeln
    end
```

قراءة نهاية العدّاد

شكل (٤ - ٢)

**ملاحظة :** عندما يعمل البرنامج السابق سوف يستقبل قيمة نهاية العدّاد f ويبدأ في طباعة العبارة المطلوب تكرارها على نفس السطر . وفي هذه الحالة يفضل استخدام عبارة القراءة readln بدلاً من read حيث أنها تسمح بالانتقال إلى سطر جديد بعد القراءة . وهى تكافئ استخدام العبارة writeln مع العبارة read بهدف الانتقال إلى سطر جديد بعد القراءة .

وهذا هو البرنامج والتنفيذ :

مثال (٤ - ٣) :

البرنامج

```
var f: integer;
begin
  readln(f);
  for i:=1 to f do
  begin
    writeln('sorry..say again');
    writeln
  end
end.
```

القراءة على سطر مستقل

التنفيذ

```
Running
3 ← البيان المدخل
sorry..say again
sorry..say again
sorry..say again
```

النتيجة

/

```
Running
5 ← البيان المدخل
sorry..say again
sorry..say again
sorry..say again
sorry..say again
sorry..say again
```

النتيجة

شكل (٤ - ٣)

### (٤ — ٣) خصائص الحلقة التكرارية : for

- يجوز استخدام متغير العداد لأداء وظيفة أخرى بداخل الحلقة التكرارية مثل طباعة الأرقام المسلسلة :

**for x:= 1 to 5 do write(x:4)**

بموجب هذه الحلقة يقوم البرنامج بتكرار العبارة `write(x:4)` خمس مرات وفي كل مرة يطبع قيمة متغير العداد `x` في أربعة خانات . فيكون ناتج هذا البرنامج عبارة عن الأرقام من 1 إلى 5 متجاورة أي :

1   2   3   4   5

- يتزايد متغير العداد بخطوة مقدارها 1 عند كل تكرار لكنه قد يبدأ من أي قيمة ابتدائية سالبة أو موجبة :

**for x:= -5 to 5 do write(x:3)**

تقوم هذه الحلقة بطبع الأرقام من (-5) إلى (+5) متجاورة في خانات ثلاثية كالتالي :

(  
-5 -4 -3 -2 -1 0 1 2 3 4 5

- يتزايد عدّد الحلقة التكرارية بخطوة مقدارها 1 ، لذلك فإذا أردنا طباعة الأعداد الزوجية نطلب ذلك شيئاً من المهارة .

**for x:= 1 to 50 do writeln(n\*2:3)**

يطبع هذا البرنامج الأعداد الزوجية متتالية تحت بعضها البعض من 2 إلى 100 .

**for x:=1 to 50 do writeln(n\*2-1:3)**

أما هذا البرنامج فيطبع الأعداد الفردية من 1 إلى 99 . وكذلك يمكن كتابته بالطريقة الآتية :

**for x:=0 to 49 do writeln(n\*2+1:3)**

(٤ — ٤) الحلقة التكرارية بخطوة سالبة :

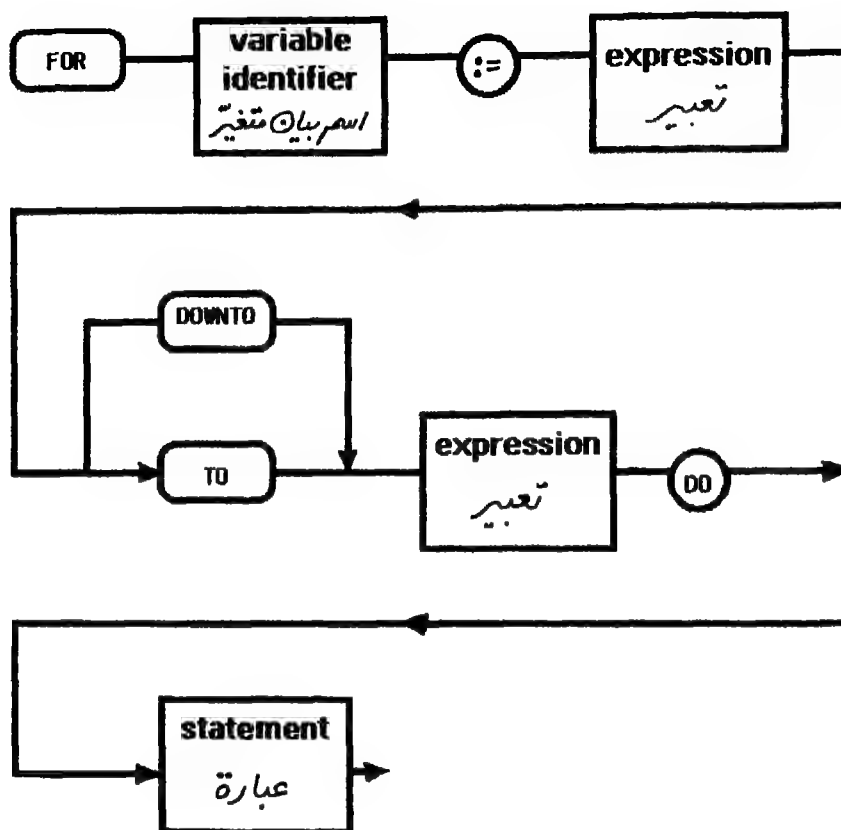
لاحظنا أن الحلقة التكرارية التي تعاملنا معها النوا كان العداد يتزايد فيها تزايداً مطرداً . لذلك يقال لها أنها حلقة ذات خطوة موجبة .

ويجوز أن تكون الخطوة سالبة أيضاً بمعنى أن تناقص قيمة متغير العداد في كل مرة كالمثال الآتي :

**for x:=100 downto 0 writeln(x:3)**

فهنا تناقص خطوة العداد من 100 حتى نصل إلى الصفر بموجب الكلمة **downto** التي حلت محل كلمة **to** في الصيغة السابقة للعبارة for .

ويمكن التعبير عن الحلقة التكرارية بسوعيا (ذات الخطوة الموجبة) وذات الخطوة السالبة بالرسم كما في شكل (٤ — ٤) .



شكل (٤ - ٤)

مثال (٤ — ٥) عمولة مندوب مبيعات :

يحصل مندوب مبيعات إحدى الشركات على عمولة بمعدل ١٢,٥٪ من سعر المبيعات التي لا يتعدى ثمنها ١٠٠ جنيه ، ويزيد هذا المعدل إلى ١٥,٥٪ على المبيعات التي يتعدى ثمنها المائة .

والبرنامج التالي يقوم بطبع العمولة التي يحصل عليها من أى عملية وحتى ٢٠٠ جنيه كحد أقصى لسعر البيع (وبالطبع يمكن تغيير هذا الحد الأقصى بحسب نوع السلعة المباعة) .

```

البرنامج :
program commissiontable(output);
const lowerrate = 0.125; higherrate = 0.155;
topoflowerrange=100;
var sale : integer;
begin
    writeln('sale    commission'); writeln;
    for sale := 1 to topoflowerrange do
        writeln(sale:4, sale*lowerrate:13:2);
    for sale := topoflowerrange+1 to 200 do
        writeln(sale:4, sale*higherrate:13:2)
end.

```



sale	commission	التنفيذ :
1	0.12	
2	0.25	
3	0.37	
4	0.50	
5	0.62	
.	.	
.	.	
.	.	
99	12.38	
100	12.50	
101	15.65	
102	15.81	
.	.	
.	.	
.	.	
.	.	
199	30.84	
200	31.00	

شكل (٤ - ٥)

(٤ - ٥) الاختيار بين البدائل (If-statement) :

من « خصائص الكمبيوتر » أنه يستطيع اتخاذ القرارات بمعنى الاختيار بين البدائل المتعددة والخروج بالقرار النهائى . وهذه الخاصية هي إمكانية من إمكانات اللغة التي نستخدمها لبرمجة الكمبيوتر .

ونحن في غنى عن القول بأن علينا أن نخبر الكمبيوتر بكيفية اتخاذ القرار وطريقة المقارنة بين البدائل إذا أردناه أن يتخذ لنا قرار ما .

والعبارة التي تمدنا بها لغة باسكال لهذا الغرض هي العبارة if ولنر هذا

المثال :

**if age < 18 then**

**write('age is less than required')**

إن العبارة if في هذا المثال تفصح عن معناها فهي تقول ببساطة : « إذا كان العمر أقل من ١٨ عاماً فاطبع الرسالة : السن أقل من المطلوب » والعمر أو السن يعبر عنه بالمتغير العددي age .

ولنر هذا المثال :

**if c > 3 then**

**fine := fine\*2**

وهذا قرار تختص به الشرطة حيث أنه يبحث الحالة عندما تزيد عدد مرات ارتكاب المخالفة عن ثلاثة مرات . فهنا تمثل عدد المرات بالمتغير c ، فإذا زادت قيمة المتغير c عن ثلاثة قام البرنامج بمضاعفة قيمة المخالفة (fine) بموجب العبارة :

**fine := fine\*2**

فإذا كانت قيمة المتغير fine هي ٥٤ مثلاً فإنه بموجب هذه العبارة يتم ضرب هذه القيمة في الرقم ٢ وتخصص القيمة الناتجة لنفس المتغير fine . وهذا مثال آخر نتعامل فيه مع أرقام مجردة :

**if t > 100 then**

**t := t - 0.10\*t**

في هذا المثال أيضاً يتم تغيير القيمة المخزنة في المتغير  $t$  إذا زادت عن المائة .  
والتغيير المطلوب هو طرح العُشر من قيمة المتغير  $t$  .  
لعلنا لاحظنا في كل الأمثلة السابقة أن العبارة  $if$  تتبع صيغة معينة هي :



شكل (٤ - ٦)

أي أنها تتكون من جزئين الأول هو الشرط والثاني هو النتيجة ، والشرط عادة يكون تعبيراً منطقياً مثل :

$age < 18$

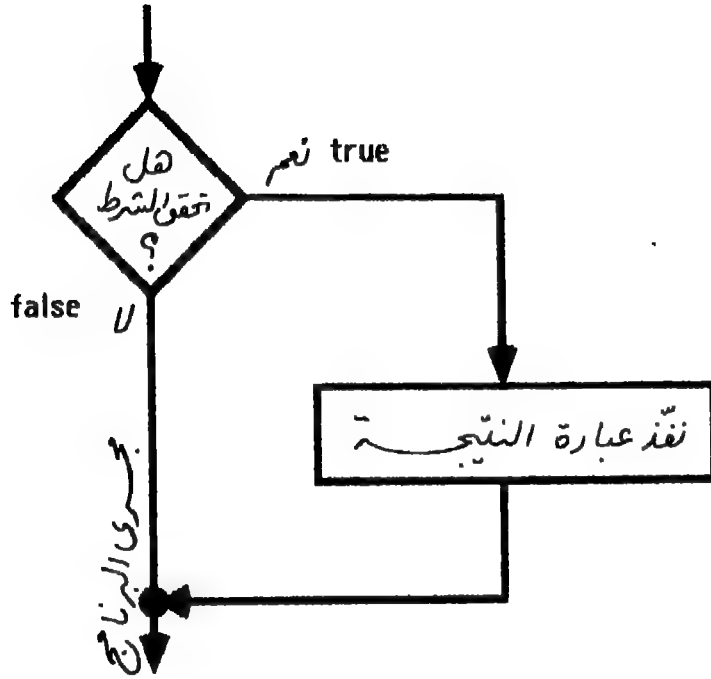
$t > 100$

$c > 3$

أما النتيجة فهي أي عبارة من عبارات اللغة التي من شأنها أن تعبر عن النتيجة المطلوبة .

ولعل بعضنا الآن يتساءل : « ماذا يفعل البرنامج إذا لم يتحقق الشرط الوارد في البرنامج » ؟ وهذا سؤال هام جداً . فالعبارة الشرطية تعترض مجرى البرنامج لاختبار الشرط المقصود فإذا تحقق الشرط تحوّل مجرى البرنامج إلى مجرى جانبي لتنفيذ العمليات الواردة في عبارة النتيجة . أما إذا لم يتحقق الشرط فإن البرنامج يستمر في تسلسله العادي دون الالتفات لعبارة النتيجة .

أى أنه يمكن التعبير عن منطق العبارة if بالرسم التالى :



شكل (٤ - ٧)

ولكن العبارة الشرطية لها بقية !

ففى بعض المواقف قد نحتاج للتعامل مع كل من النتيجة الموجبة (true) والنتيجة السالبة (false) كل بطريقة مختلفة قبل أن ننضم إلى مجرى البرنامج .

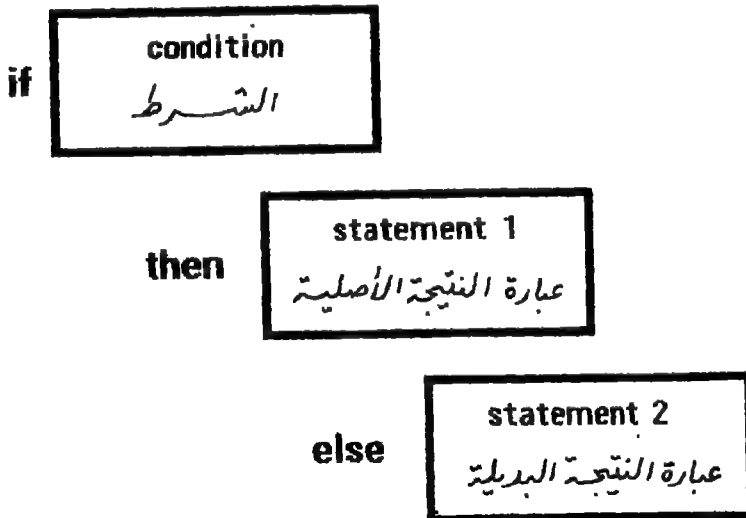
فمثلاً : قد نحتاج أن نضيف إلى المثال الأول إضافة ما مثل : « إذا كان العمر أقل من ١٨ فاطبع الرسالة :

(السن أقل من المطلوب) وإلا فاطبع الرسالة :  
(مستوف لشروط السن) .

يمكن التعبير عن هذا الشرط المركب بالعبارة الشرطية الكاملة كالآتي :

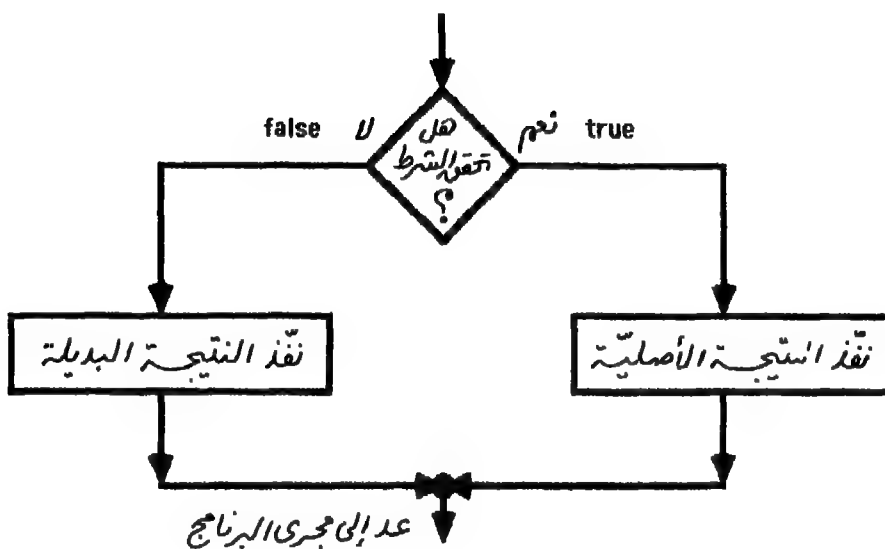
```
if age < 18 then
    write('age is less than required')
else
    write('age satisfied')
```

لقد ظهرت هنا كلمة جديدة في العبارة الشرطية هي else بمعنى « وألا »  
والجزء الذى يتبع else يسمى عبارة النتيجة البديلة (وتسمى النتيجة الأخرى  
بالنتيجة الأصلية) أى أن صيغة العبارة الشرطية الكاملة كالآتي :



شكل (٤ - ٨)

وما يحدث في البرنامج بالفعل أن المجرى الأصلي للبرنامج يتفرع إلى فرعين يختص أحدهما بمعالجة النتيجة الأصلية والآخر بمعالجة النتيجة البديلة ثم يجتمعان مرة أخرى في مجرى . البرنامج الأصلي كما في الشكل (٤ - ٩) .



شكل (٤ - ٩)

ولا بأس أن يحتوى البرنامج على أكثر من عبارة شرطية تأتي متتالية وراء بعضها البعض وهي في هذه الحالة تشبه سلسلة الاختبارات التي يمر بها الطالب المتقدم للكليات العسكرية !

(٤ - ٦) تطبيقات على العبارة الشرطية :

مثال (٤ - ٦) مقارنة ثلاثة أعداد :

يقوم هذا البرنامج بقراءة ثلاثة أرقام وطباعة أكبر رقم . ومتروك لك تتبع منطق البرنامج علماً بأن الأرقام الثلاثة تستقبل في المتغيرات المعبرة عن أسمائها :

**first, second, third**

كما تم استخدام المتغير largest كوعاء لكي نضع فيه الرقم الأكبر ، والمتغير largestsofar كوعاء مؤقت أثناء المقارنات وهذا الاسم يعنى « الأكبر حتى الآن ... » .

```
program largest(input,output);
var first, second, third, largestsofar, largest : real
begin
  read(first, second, third);
  if first > second then
    largestsofar := first
  else
    largestsofar := second;
  if largestsofar > third then
    largest := largestsofar
  else
    largest := third;
  writeln('biggest number is ', largest)
end.
```

مقارنة ثلاثة أعداد

شكل (٤ - ٦)

قبل أن نتقل إلى المثال الثانى دعنا نتحدث قليلاً عن الفصولات المنقوطة

(semicolons) وهى أمر قد يشغل بال من كان جديداً على التعامل مع لغة باسكال .

لعلنا لاحظنا أن الفاصلة المنقوطة لم تظهر خلال العبارة الشرطية إلا عند نهايتها أى بعد النتيجة البديلة . ذلك لأن العبارة الشرطية سواء كانت بسيطة أو كاملة تعتبر عبارة واحدة متصلة رغم احتوائها على عبارات كاملة بداخلها . وهذه الملاحظة من الأهمية بمكان . فقد ذكرنا من قبل أنه لا بأس من أن نفرط فى استخدام الفاصلة المنقوطة حيث لا يتطلب الموقف ذلك مثل استخدامها مع العبارة الأخيرة فى البرنامج . ولكن هذا لا يعنى بحال أن تأتى الفاصلة المنقوطة فى منتصف عبارة ما ، فهذا يفسد معنى العبارة . لذلك لا يجوز أن يتخلل العبارة الشرطية بأجزائها أية فاصلة منقوطة حتى يستطيع الكمبيوتر التعامل معها ككل موحد .

مثال (٤ — ٧) المزيد من المقارنات :

يقوم هذا البرنامج باستقبال أربعة أرقام . سوف نطلق على الرقم الأول الاسم standard بمعنى الرقم القياسى وسوف يتم استقبال الأرقام الثلاثة الأخرى واحداً بعد الآخر فى وعاء واحد (متغير واحد) يحمل الاسم next . وبعد استقبال كل رقم من الأرقام الثلاثة يتم اختبار الفرق بينه وبين الرقم القياسى فإذا كان هذا الفرق أصغر من العشر فإن قراءة العداد numberclose (وهو متغير صحيح) تزداد بمقدار 1 بموجب العبارة :

**numberclose := numberclose + 1;**

(بمعنى أضف واحداً إلى قيمة المتغير numberclose وخصص القيمة الجديدة لنفس المتغير) .

وبذلك فإن هذا البرنامج يقوم بحصر عدد الأرقام التى تقترب قيمتها من قيمة الرقم القياسى standard .



فإذا كان لدينا الأرقام الأربعة الآتية :

**3.156    3.051    3.152    3.091**

فإن خرج البرنامج يكون كالاتي :

**2 values are near the standard**

```

program tolerance(input,output);           البرنامج
var standard, next : real;  numberclose : integer;

begin
  numberclose := 0;
  read(standard);
  read(next);
  if abs(standard - next) < 0.1 then
    numberclose := numberclose + 1;
  read(next);
  if abs(standard - next) < 0.1 then
    numberclose := numberclose + 1;
  read(next);
  if abs(standard - next) < 0.1 then
    numberclose := numberclose + 1;
  writeln(numberclose,
           ' values are near the standard')
end.

```

المراقبة الرقمية للإنتاج

**شكل (٤ - ١١)**

ويستخدم هذا البرنامج في المراقبة الرقمية للإنتاج حيث تقارن أبعاد المنتجات بالأبعاد القياسية المخزنة في ذاكرة الكمبيوتر .

ولعلنا لاحظنا في هذا البرنامج تكرار عبارة قراءة المتغير والعبارة الشرطية .

وبالطبع هناك طريقة أفضل لأداء نفس البرنامج وذلك بوضع كل العبارات المتكررة بداخل حلقة تكرارية فيصبح البرنامج كما في شكل (٤ - ١٢) .  
وبتغيير الحد النهائي لمتغير العداد count يمكن التحكم في عدد مرات التكرار كيفما نشاء (سوف يتم التعرض لهذه المنشئات في الباب الخامس بالتفصيل) .

```
program tolerance2(input,output);  
var standard, next : real;  
    numberclose, count : integer;
```

البرنامج

```
begin
```

```
    numberclose := 0;  
    read(standard);
```

المراقبة الرقمية للإنتاج ؟

```
    for count := 1 to 3 do  
    begin
```

```
        read(next);  
        if abs(standard - next) < 0.1 then  
            numberclose := numberclose + 1
```

```
    end;
```

```
writeln(numberclose, ' values are near the standard.')
```

```
end.
```

شكل (٤ - ١٢)

مثال (٤ - ٨) اختبار السنة الكبيسة :

يحتوى البرنامج على متغيرين الأول هو year وهو متغير صحيح يمثل السنة أما الآخر فهو leap وهو متغير منطقي لتمثيل حالتى السنة : « كبيسة أو غير كبيسة » ، فهو يأخذ القيمة الصحيحة true إذا كانت السنة كبيسة ويأخذ القيمة غير الصحيحة false إذا لم تكن كذلك .

والشرط الوارد في البرنامج لاختبار السنة الكبيسة أنه إذا كان العدد المعبر عن السنة يقبل القسمة على 4 ولا يقبل القسمة على 100 في نفس الوقت تكون السنة كبيسة ويأخذ المتغير leap القيمة الصحيحة true . ولكن هناك حالة أخرى تصبح فيه السنة كبيسة وهي أن تقبل القسمة على 400 . ولذلك نلاحظ أن التعبير المنطقي الذي جاء في البرنامج تعبير مركب ويتكون أساساً من شرطين بينهما المؤثر المنطقي OR .

فإذا تتبعنا البرنامج سطرًا بسطر نجد أنه يبدأ بالإعلانات ثم يلي ذاك جسم البرنامج الرئيسي بعد البداية Begin .

تبدأ عمليات المعالجة بقراءة السنة بالعبارة read يليها طبع سطر خال بواسطة العبارة writeln . ورغم أن هذه الخطوة لا تمت إلى موضوع البرنامج بصلة ، لكنها عادة حميدة أن تتبع العبارة read بسطر خال حتى لا تختلط الكتابة على الشاشة ولكي تستطيع تمييز المدخلات التي تُغذى بها البرنامج من المعلومات التي يطبعها الكمبيوتر على الشاشة .

السطر التالي هو تخصيص التعبير المنطقي للمتغير leap وهو يقول :

« إذا كانت السنة تقبل القسمة بدون باق على العدد 4 وكانت لا تقبل القسمة بدون باق على العدد 100 .  
أو

إذا كانت السنة تقبل القسمة بدون باق على العدد 400 فإن المتغير leap يأخذ القيمة true » .

يلي ذلك العبارة الشرطية if وقد جاءت هنا بسيطة للغاية :

**if leap then.....**

بمعنى أنه إذا كان المتغير leap يحتوي على القيمة true .. فاطبع كذا وكذا ...  
كما ظهر الجزء المكمل للعبارة else متضمناً النتيجة البديلة عندما لا يكور

المتغير leap صحيحاً .

```
program leapyear(input,output);
```

البرنامج

```
var year:integer;
    leap:boolean;
```

```
begin
```

```
  read(year);
```

```
  writeLn;
```

```
  leap:=(year mod 4=0) and (year mod 100<0)
```

```
        or (year mod 400=0);
```

```
  if leap then writeLn(year,' is a leap year ');
```

```
  else writeLn(year,' is not a leap year');
```

```
end.
```

اختبار السنة الكبيسة

التنفيذ لسنوات مختلفة :

الرقم المدخل  
1988  
1988 is a leap year  
النتيجة...  
سنة ١٩٨٨ سنة كبيسة

2000  
2000 is a leap year

>

1986  
1986 is not a leap year

شكل (٤ - ١٣)

ويمكنك اختبار النواتج التي حصلنا عليها من البرامج باختبار السنوات المدخلة وهي 1988 ، 2000 ، 1986 وذلك بإجراء الحسابات على آلة حاسبة أو يدوياً لمعرفة ما إذا كانت هذه السنوات كبيسة أم لا .

وهناك طريقة مثلى لاختبار مجموعة كبيرة من الأرقام الممثلة للسنوات وذلك باستخدام الحلقات التكرارية بتعديل البرنامج تعديلاً طفيفاً ليكون كما هو موضح في شكل (٤ - ١٤) .

```
program leapyear(input,output):
```

```
var year:integer;
```

```
    leap:boolean;
```

```
begin
```

```
    for year:=1970 to 1990 do
```

```
    begin
```

```
        leap:=(year mod 4=0) and (year mod 100<>0)
```

```
            or (year mod 400=0);
```

```
        if leap then writeln(year,' is a leap year')
```

```
            else writeln(year,' is not a leap year')
```

```
        end
```

```
    end.
```

البرنامج

اختبار السنة الكبيسة

شكل (٤ - ١٤)

وعند تنفيذ هذا البرنامج فإن الحلقة التكرارية التي تبدأ بالرقم 1970 وتنتهي بالرقم 1990 هي التي تغذي البرنامج بالأرقام الممثلة للسنوات ، وعلى ذلك يكون ناتج هذا البرنامج كما هو موضح في شكل (٤ - ١٥) .

### تنفيذ البرنامج لعدة سنوات

1970 is not a leap year  
1971 is not a leap year  
1972 is a leap year  
1973 is not a leap year  
1974 is not a leap year  
1975 is not a leap year  
1976 is a leap year  
1977 is not a leap year  
1978 is not a leap year  
1979 is not a leap year  
1980 is a leap year  
1981 is not a leap year  
1982 is not a leap year  
1983 is not a leap year  
1984 is a leap year  
1985 is not a leap year  
1986 is not a leap year  
1987 is not a leap year  
1988 is a leap year  
1989 is not a leap year  
1990 is not a leap year

### شكل (٤ - ١٥)

#### (٤ - ٧) العبارة الشرطية المتعددة النتائج :

استخدمنا العبارة الشرطية البسيطة ذات النتيجة الواحدة وكذلك العبارة الشرطية الكاملة ذات النتيجةين الأصلية والبديلة .

والواقع أن نتيجة العبارة الشرطية بنوعها يمكن أن تتعدد ، بمعنى أنه يمكن أن يترتب على تحقق الشرط (أو عدم تحققه) عدة نتائج متتابعة . في هذه الحالة

فإننا سوف نحتاج أن نفصل هذه النتائج المتابعة عن بقية عبارات البرنامج بوضعها بين عبارتي بداية (begin) ونهاية (end) اللتين تعملان عمل الأقواس في التعبيرات الحسابية .

والبرنامج التالي تطبيق جيد للعبارة الشرطية المتعددة النتائج وهو يستقبل عددين ويطبّع مجموعهما والفرق بينهما . ويبدأ البرنامج بقراءة العدديتين وتخصيص أحدهما للمتغير larger (بمعنى الأكبر) والآخر للمتغير smaller (بمعنى الأصغر) .

بعد ذلك تتم مقارنة العددين للتأكد من أن الأصغر يحتل المتغير smaller والأكبر يحتل المتغير larger فإن لم يكن كذلك ، فتجرى عملية تبديل لقيم المتغيرات .

وعملية التبديل تتم باستخدام المتغير الوسيط temporary ، تماماً كما لو كان لدينا كوباً من الشاي وآخر من القهوة فإننا إذا أردنا تبديل محتويات الكوبين فلا بد لنا من الاستعانة بكوب ثالث لإجراء عملية التبديل .

وتجرى عملية التبديل كنتيجة لتحقيق الشرط في العبارة if بموجب العمليات الثلاثة الآتية :

**temporary: = larger;**

**larger: = smaller;**

**smaller: = temporary;**

حيث تبدأ العملية « بتفريغ » قيمة المتغير larger في المتغير المؤقت temporary ثم تفريغ محتويات المتغير smaller في المتغير larger ثم تفريغ محتويات المتغير المؤقت في المتغير smaller .

بهذه الخطوات الثلاث تكون محتويات المتغيرين smaller ، larger قد تم تبديلهما .

مثال (٤ - ٩) :

```

program twonumbers(input,output);
var larger, smaller, temporary, sum,
    difference : integer;

begin
    read(larger, smaller);

    if larger < smaller then
    begin
        temporary := larger;
        larger     := smaller;
        smaller    := temporary
    end;

    sum          := larger + smaller;
    difference   := larger - smaller;

    writeln('the sum is ', sum);
    writeln('the difference is ', difference);
    writeln('nos. in order are ', larger, smaller)
end.

```

← العبارة الشرطية متعددة النتائج

شكل (٤ - ١٦)

(٤ - ٨) الاختيار بين البدائل المتعددة case :

العبارة الشرطية التي تعاملنا معها حتى الآن تحتوي دائماً على بديلين فقط والاختيار بينهما يتم وفقاً لتحقيق شرط واحد سواء كان بسيطاً كالمقارنة بين عددين أو كان شرطاً مركباً مثل شرط السنة الكبيسة .

هذان البديلان كأنهما وجهان لعملة واحدة وما أكثر ما نلتقي بهما في حياتنا اليومية مثل « الخير والشر » ، « الحرب والسلام » ، « النجاح والفشل » .



فلو أنك هبطت محطة كبيرة للسكك الحديدية مثل محطة القاهرة فسوف تدهمك أفواج المسافرين الذين يتفرعون أو ينضمون في مسالك مختلفة وعشرات القطارات المتأهبة للتحرك . وعليك في هذا الزحام الهائل أن تتحقق من شرط معين حتى تصل إلى القطار المطلوب الذى يرحل بك إلى مقصدك في سلام . إن هذا الشرط الذى تتحقق منه عادة هو رقم الرصيف وساعة القيام ، ويتحقق هذا الشرط فإنك تختار بديلاً واحداً (قطاراً واحداً) من بين عشرات البدائل الأخرى التى ربما تنطبق شروطها على مسافرين آخرين .

وأحد الوسائل التى يمكننا بها برجمة مثل هذه الحالة هى استخدام عدد من العبارات الشرطية مساو لعدد الحالات المختلفة ولكن يا له من عمل شاق .

ولأن هذا التطبيق هو أحد التطبيقات الهامة للكمبيوتر فإن لغة باسكال تمدنا بوسيلة فعالة في مثل هذه الحالات وهى العبارة المركبة case . ومن سبق له دراسة لغة بيسك من قبل فإنه ولا بد قد تعرّف إلى العبارة ON GOTO التى تؤدى نفس الغرض :



#### مثال (٤ - ١٠) :

ولنبداً بهذا المثال عن ماكينة المشروبات التي تغذيها بقطع العملة المختلفة فتقوم الماكينة بحساب قيمة ما دخل إليها من نقود . وبالضغط على زر ما تستطيع اختيار ما تشاء من المشروبات أو المأكولات فتقوم الماكينة في هذه الحالة بإخراج الصنف المطلوب وكذلك إخراج باقي الحساب في وعاء صغير مخصص للنقود .

كيف نعمل هذه الماكينة ؟

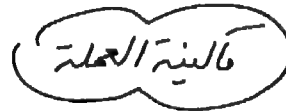
إن المبدأ الرئيسى هو مقارنة الأوزان المختلفة لقطع العملة وتحديد قيمة كل قطعة تبعاً لوزنها .

وسوف نفترض أن العملات المستخدمة هي :

- العملة ذات القيمة ١٠ قروش وتزن ٣٥ جراماً .
- العملة ذات القيمة ٥ قروش وتزن ١٦ جراماً .
- العملة ذات القيمة ١ قروش وتزن ٩ جراماً .
- العملة ذات القيمة  $\frac{1}{4}$  قروش وتزن ٧ جراماً .

في هذه الحالة يمكن استخدام العبارة case في برنامج باسكال كالتالى :

```
read (weight);
case weight of
35: amount:=amount+10;
16: amount:=amount+ 5;
9 : amount:=amount+ 1;
7 : amount:=amount+0.5;
end
```



#### شكل (٤ - ١٧)

فإذا كان وزن العملة المدخلة هو ٣٥ جراماً فإن البرنامج يضيف مقدار ١٠ قروش إلى المبلغ المخزن في المتغير amount وإذا كان الوزن هو ١٦ جراماً يضيف البرنامج مقدار ٥ قروش وهكذا .

ويمثل المتغير amount جملة القيمة المالية المدخلة إلى الماكينة والتي على أساسها يتم الشراء واستعادة ما تبقى من نقود لدى الماكينة .

والمتغير weight يمثل وزن العملة المدخلة بالجرام . وسوف نعود إلى هذا البرنامج مرة أخرى لنستكمل بقية خطواته عندما نمضي في عبارات باسكال أبعد من ذلك . لكن ما يهمنا في الوقت الحالى هو الصيغة التي نكتب بها العبارة case والتي يمكن تقنيها كالتالي :

case

متغير الاختيار

of

العبارة  
:  
:  
:  
:  
:  
:

end

شكل (٤ - ١٧)

فالعبارة case تمثل برنامجاً صغيراً يقع بداخل البرنامج الرئيسي ، وهي تبدأ بكلمة case يعقبها متغير الاختيار الذي تتعدد النتائج تبعاً لقيمته (وهو المتغير

weight في المثال السابق) يليه كلمة of .

بعد ذلك تتالى العبارات التى تصف منطق الاختيار والتى تبدأ كل منها « بعنوان » رقمى يمثل قيمة معينة لمتغير الاختيار ويتبع العنوان نقطتان (١٠) ثم العبارة المطلوب تنفيذها إذا وقع الاختيار على العنوان الذى يبدأ به السطر . وتنتهى العبارة المركبة case بكلمة end لتسلم دفعة التنفيذ بالبرنامج الرئيسى . ويمكن أن يضم السطر الواحد أكثر من عنوان .

مثال (٤ - ١١) :

المثال التالى تستخدم فيه العبارة case لطبع إحدى ثلاث رسائل الرسالة الأولى هي :

Low season rate « موسم ذو معدل منخفض »

Mid season rate « موسم ذو معدل متوسط »

Peak season rate « موسم ذو معدل عال »

وهذه الرسائل الثلاثة تناظر المواسم التجارية التى تدور مع أشهر السنة ، فالشهور يناير وفبراير ونوفمبر وديسمبر تناظر الموسم المنخفض . أما الشهور مارس وأبريل ومايو وأكتوبر فهى تناظر الموسم المتوسط ، والشهور يونيو ويوليو وأغسطس وسبتمبر فهى تمثل الموسم المرتفع المعدل .

وهذه هي فقرة البرنامج المعبرة عن هذا المنطق :

```
read(month);
case month of
  1,2,11,12 : writeln('low season rate');
  3,4,5,10  : writeln('mid season rate');
  6,7,8,9   : writeln('peak season rate')
end
```

متغير الاختيار

شكل (٤ - ١٨)

أما متغير الاختيار فلا يكون بالضرورة ثابتاً عددياً بل من الجائز استخدام التعبيرات الحسابية والمنطقية أيضاً .

مثال (٤ - ١٢) :

في المثال التالي نستخدم تعبيراً حسابياً للتعبير عن دخل الفرد الذي تتحدد على أساسه الضريبة التصاعدية .

(ملاحظة : هذا المثال لمجرد التدريب ولا يمثل أى قانون حقيقى للضرائب) ولنعتبر الحالات الآتية على سبيل المثال :

- إذا كان الدخل بين 1000 ، 0 كانت نسبة الضريبة 0 %
- إذا كان الدخل بين 2000 ، 1000 كانت نسبة الضريبة 5 %
- إذا كان الدخل بين 3000 ، 2000 كانت نسبة الضريبة 7.5 %
- إذا كان الدخل بين 4000 ، 3000 كانت نسبة الضريبة 10 %
- إذا كان الدخل بين 5000 ، 4000 كانت نسبة الضريبة 12 %

ولبرمجة هذه الحالات نستخدم متغيراً صحيحاً هو income يمثل الدخل وآخر tax يمثل نسبة الضريبة . وبلاستفادة من خاصية القطع truncation لعملية القسمة الحقيقية يمكن كتابة البرنامج التالى :

```
program tax(input,output):
var tax,income:real;
begin
  read (income);
  case trunc(income/1000) of
    0:tax:=0;
    1:tax:=5;
    2:tax:=7.5;
    3:tax:=10;
    4:tax:=12;
  end;
  writeln;
  writeln('tax to pay: ',income*tax*100)
end.
```

البرنامج

دالة القطع

حساب الضرائب

شكل (٤ - ١٩)

ونلاحظ في هذا البرنامج أن أى عدد أقل من 1000 يعطى صفراً عند قسمته على 1000 . وإذا كان العدد أكبر من 1000 بأى نسبة فإنه يعطى الرقم 1 وهكذا ....

مثال (٤ - ١٣) :

أما في هذا المثال فنستخدم فيه التعميرات المنطقية للتعبير عن نفس مضمون البرنامج شكل (٤ - ١٠) .

ولكن باستخدام منطق جديد هذه المرة :

```

case first > second of
  true : largestsofar := first;
  false: largestsofar := second
end;

case largestsofar > third of
  true : largest := largestsofar;
  false: largest := third
end;

```

مقارنة ثلاثة أعداد

```

writeln('largest value is ', largest)

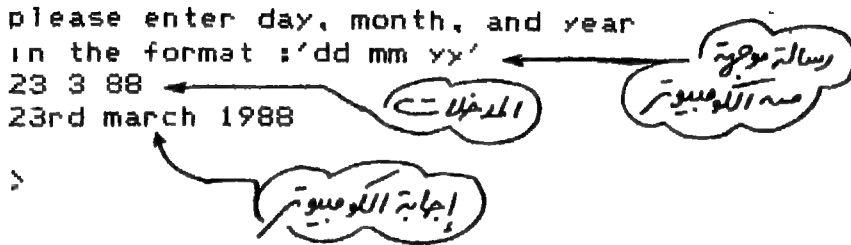
```

شكل (٤ - ٢٠)

ومن الجدير بالذكر أنه قد يتطلب البرنامج ، عند قيمة معينة لمتغير الاختيار ، تنفيذ أكثر من عبارة . وفي هذه الحالة فقد علمنا من قبل أن العبارتين begin ، end تستخدمان كما تستخدم الأقواس في الحسابات لحصر مجموعة معينة من العبارات وعزلها عن متسلسل البرنامج الأصلي .

مثال (٤ - ١٤) :

لنتصور برنامجاً يعمل بالصورة الآتية كما في الشكل التالي :



شكل (٤ - ٢١)

لقد بدأ البرنامج بأن سألنا إدخال تاريخ اليوم معبراً عنه بالفورمات dd mm yy . أى أن كل من اليوم والشهر والسنة يتم التعبير عنه بعدد مكون من رقمين (على الأكثر) . لذلك قمنا بإدخال الأرقام الموضحة وهي تتكون من :

- العدد 23 للتعبير عن اليوم .
- العدد 3 للتعبير عن الشهر (مارس) .
- العدد 88 للتعبير عن العام (١٩٨٨) .

فكانت إجابة الكمبيوتر هي :

**23rd march 1988**



لقد استبدل الشهر رقم ٣ بالاسم « مارس » وأضاف ١٩ إلى العدد 88 ليكون 1988 كما نلاحظ أنه أضاف حرفين (rd) للعدد 23 المعبر عن اليوم ، حتى يمكن قراءة العدد على النحو التالي :

**twenty third of march 1988**

فلنر معاً كيف يكون الحال مع تاريخ آخر :

please enter day, month, and year  
in the format : 'dd mm yy'  
6 10 73  
oth october 1973 ←

>

السادس من أكتوبر ١٩٧٣

شكل (٤ - ٢٢)

إن هذا اليوم هو السادس من أكتوبر ١٩٧٣ . ونلاحظ أنه قد وضع الحرفين th أمام الرقم 6 هذه المرة حتى تُقرأ :

**sixth of october 1973**

كيف يمكن بناء هذا البرنامج ؟

إن المطلوب من البرنامج التعرف على الشهور لاستبدال كل رقم يدل على شهر باسم هذا الشهر .

كما هو مطلوب إضافة نهايات مناسبة للأيام . فمع اليوم الأول (1) أو الحادى والعشرين (21) أو الحادى والثلاثين (31) تضاف النهاية st لتدل على كلمة first . ومع الأيام (2) ، (22) تضاف النهاية nd لتدل على كلمة

. second

ومع الأيام (3) ، (23) تضاف النهاية rd لتدل على كلمة third .

أما عدا ذلك فتضاف النهاية المعتادة th .

ومع العام فالمطلوب شيء بسيط وهو طبع بداية للعدد الدال على العام وهو

العدد (19) .

وهذا هو البرنامج :

البرنامج

```

program date(input,output);
var d,m,y:integer;
begin
  writeln('please enter day, month, and year');
  writeln('in the format : ''dd mm yy''');
  read (d,m,y);
  writeln;
  case d of
    1,21,31:write(d,'st');
    2,22   :write(d,'nd');
    3,23   :write(d,'rd');
    4,5,6,7,8,9,10,11,12,13,14,15,16:write(d,'th');
    17,18,19,20,24,25,26,27,28,29,30:write(d,'th');
  end;
  case m of
    1:write(' January');
    2:write(' february');
    3:write(' march');
    4:write(' april');
    5:write(' may');
    6:write(' June');
    7:write(' July');
    8:write(' august');
    9:write(' september');
    10:write(' october');
    11:write(' november');
    12:write(' december');
  end;
  writeln(' 19',y);
end.

```

المدخلات ←

طباعة اليوم } ←

طباعة الشهر } ←

طباعة العام ←

شكل (٤ - ٢٣)

نلاحظ في هذا البرنامج ما يلي :

١ — إن السطر الذى يطبعه الكمبيوتر والذى يعبر عن التاريخ يُطبع على ثلاث مراحل فالיום يطبع مع الروتين case المخصص لاختبار البدائل المختلفة للأيام .

أما الشهر فيطبع أثناء الروتين case الثانى المخصص لاختبار البدائل المختلفة للشهور .

أما العام فيطبع في نهاية البرنامج .

ولأن عملية طباعة سطر واحد تتم على ثلاث مراحل لذلك فإننا قد استخدمنا العبارة write بدلاً من writeln عند طباعة اليوم والشهر وذلك حتى لا يتم الانتقال إلى السطر التالى .

أما عند طباعة العام فقد استخدمنا writeln حيث وصلنا إلى نهاية السطر .

٢ — المتغيرات المستخدمة هنا هى :

d : للتعبير عن اليوم

m : للتعبير عن الشهر

y : للتعبير عن العام

٣ — فى الروتين case الأول نلاحظ أن الحالة التى تطبع فيها النهاية 'th' قد قسمت إلى حالتين ، وهذا لمجرد تنظيم شكل البرنامج لكنه يجوز أن يتضمن السطر الواحد أى عدد من البدائل .

٤ — عند إدخال قيم هذه المتغيرات يراعى ترك مسافة خالية بين كل عدد وآخر ثم الضغط على الزر ENTER (أو RETURN) فى نهاية الإدخال .

والعدد المدخل يتكوّن من رقمين على الأكثر بمعنى أن شهر مارس مثلاً يمكن كتابته 03 ويجوز كتابته أيضاً 3 .

وهذه بعض نتائج لتنفيذ البرنامج :

تنفيذ البرنامج

please enter day, month, and year  
in the format : 'dd mm yy'

06 10 73

6th october 1973

التاريخ المدخل

اجابة الكمبيوتر

تنفيذ

>

please enter day, month, and year  
in the format : 'dd mm yy'

21 03 88

21st march 1988

تنفيذ

>

please enter day, month, and year  
in the format : 'dd mm yy'

02 01 88

2nd january 1988

تنفيذ

>

شكل (٤ - ٢٤)

(٤ - ٩) الحلقات التكرارية المشروطة **Conditional loops** :

عرفنا من قبل الحلقة التكرارية for التي تستخدم لتكرار تنفيذ عملية بعينها

عدداً محدوداً من المرات .

وعرفنا العبارة الشرطية if التي بموجبها يمكن تحويل مسار البرنامج إلى مجرى معين وفقاً لتحقيق (أو عدم تحقق) شرط معين .

كما عرفنا العبارة الشرطية متعددة البدائل case التي بموجبها يتفرع البرنامج إلى عدة مسالك كل وفقاً لشرط معين .

وفي هذه الفقرة نتعرف بعبارتين جديدتين للتكرار المشروط هما العبارتان **repeat** ، **while** . وتميز هاتان العبارتان باحتوائهما على عمليتي الشرط والتكرار معاً .

#### (٤ — ٩ — ١) عبارة repeat-until :

تفيدنا هذه العبارة عندما نريد من البرنامج أن يكرر عملية معينة حتى يتحقق شرط ما ، تماماً كما نطلب من شخص ما أن يستمر في تعبئة وعاء ما حتى يمتلئ . فعملية تعبئة الوعاء عملية متكررة لكنها مرهونة بتحقيق شرط امتلاء الوعاء .

وفي اللغة العادية نقول : « املأ هذا الوعاء حتى يمتلئ » .

وفي لغة باسكال نقول :

**repeat.....until.....**

بمعنى كرّر هذه العملية حتى يتحقق شرط معين .

#### مثال (٤ — ١٥) قطار في موعد الغداء :

يريد أحد رجال الأعمال أن يستقل قطاراً من الاسكندرية إلى القاهرة في حدود الساعة ١٢ ظهراً بحيث يستغل فترة تواجده بالقطار في تناول طعام الغداء .

ولو تصورنا أنه سوف يستعين بالكمبيوتر للبحث عن القطار المناسب بسرعة ، سواء بنفسه أو عن طريق موظف الاستعلامات في محطة السكة الحديد ، فإننا يمكن أن نتصور شكل البرنامج الذى يؤدي العمل . إن المطلوب من البرنامج هو « موعد أول قطار بعد الثانية عشرة » ... هذا هو الشرط .

أما عملية البحث فهي تتم خلال مواعيد القطارات كلها التي عادة ما تكون مخزنة في « ملف » (file) يقوم الكمبيوتر بالقراءة منه آلياً حتى يعثر على الموعد المطلوب .

ولو افترضنا أن جدول المواعيد يحوى على الأرقام التالية :

وصول	قيام
1100	0806
1203	0901
1502	1155
1527	1215
1755	1450
1950	1645

فإنه بالبحث في هذا الجدول نجد أن القطار المناسب هو القطار الذى يقوم الساعة 1215 (الثانية عشرة والرابع) ويصل الساعة 1527 (الثالثة وسبعة وعشرين دقيقة بعد الظهر) .

البرنامج التالى يوضح منطق البحث في جدول المواعيد للعثور على القطار الذى يغادر بعد الساعة 1200 .

```

program lunchtrain(input,output);
const startoflunchhour = 1200;
var depart, arrive : integer;
begin
    repeat
        read(depart, arrive)
    until depart >= startoflunchhour;
    write('your train leaves at ', depart);
    writeln(' and arrives at ', arrive)
end.

```

البرنامج

قطار في موعد الغذاء

مدخلات هذا البرنامج هي :

depart	موعد القيام
arrive	موعد الوصول

كما يتضمن البرنامج الثابت المسمى startoflunchhour بمعنى « موعد بدء تقديم وجبة الغذاء » ! .

وقد تم التعبير عن الشرط المطلوب بالتعبير المنطقي :

**depart >= startoflunchhour**

أى أن موعد المغادرة يجب أن يكون بعد (أو يساوى) موعد بدء تقديم وجبة الغذاء .

أما منطق البحث فهو يبدأ بالكلمة **repeat** ، ويليهما العمليات المطلوب تكرارها وهى :

● قراءة موعد القيام والوصول .

## ● تطبيق الشروط المذكورة على موعد القيام .

وتنتهى العمليات المتكررة بموجب تحقق الشرط المطلوب وعندئذ تتولى الكلمة until والشرط المصاحب لها بإخراج البرنامج من حلقة البحث إلى التسلسل الطبيعي له حيث يقوم بطبع المطلوب .

وكما نرى أن العبارة repeat بأجزائها تمثل وحدة مستقلة بداخل البرنامج تبدأ بكلمة repeat وتنتهى بالشرط المقترن بكلمة until .

ولو افترضنا أن العبارات قد تعددت بداخل هذه الوحدة المستقلة فهى لا تحتاج لعزلها عن بقية البرنامج باستخدام begin ، end حيث أنها وحدة مستقلة بذاتها .

ويمكن التعبير عن صيغة العبارة repeat بالآتى :

**repeat**

**statements**

(العبارة المعبرة عن العملية

المطلوب تكرارها مفصولة

عن بعضها البعض بفاصلة منقوطة)

**until**

**condition**

(تعبير شرطى)

**ملاحظة :** فى هذا البرنامج تم إدخال بيانات القيام والوصول يدوياً (من لوحة الأزرار) أى إدخال ثمانية أعداد كل منهم يتكون من أربعة أرقام . وبالطبع فإن الأمور لا تجرى بهذه الطريقة وإلا كان النظر فى جدول المواعيد المعلق فى محطة القطارات أسهل بكثير .

إن البيانات يجب أن تكون مخزنة على القرص المغنطيسى (disk) فى ملف كما



ذكرنا من قبل ويتولى الكمبيوتر قراءة المواعيد آلياً بمجرد الضغط على زر تشغيل البرنامج . ولكن في المرحلة الحالية فإننا نكتفى بالبيانات المدخلة من لوحة الأزرار حتى نتقدم أبعد من ذلك في قواعد اللغة .

مثال (٤ - ١٦) تجميع بيانات تجربة معملية :

في أحد التجارب يتم تجميع القراءات في أيام متعاقبة بحيث تُجمع القراءات ، والأيام يوماً بعد يوم حتى يزيد مجموع القراءات عن حد معين . عند هذا الحد يتوقف البرنامج ليعطى النتيجة التالية :

« تعدى مجموع القراءات الحد » كذا »

وقد تم ذلك خلال مدة « كذا » يوماً .

وهذا هو البرنامج :

program countedays(input,output);

const threshold=100;

var nextreading, total :real;  
days :integer;

begin

days := 0; total := 0;

repeat

days := days + 1;

read(nextreading);

total := total + nextreading

until total > threshold;

writeln('threshold total of ', threshold,

' has been exceeded.');

writeln('this occurred after ', days, ' days.')

end.

البرنامج

بيانات تجربة معملية

شكل (٤ - ٢٦)

● في هذا البرنامج تم تعريف الحد الذي تتوقف عنده عملية التجميع بالثابت المسمى threshold وقد تم منحه القيمة العددية 100 .

● أما متغيرات البرنامج فهي :

القراءات التالية nextreading

مجموع القراءات total

عدد الأيام days

● وقد بدأ البرنامج بمنح المتغيرات القيمة صفراً لتفريغ أوعية الجمع من أى محتويات سابقة وذلك بالعبارتين :

**days:=0;total:=0;**

● بعد ذلك تم تكرار العملية الآتية بالعارة repeat :

١ — زيادة عدد الأيام بمقدار واحد :

**days:= days + 1;**

فلو بدأنا بالقيمة « صفراً » فإن ناتج هذه العملية يؤدي إلى منح المتغير days القيمة 1 .

(فالعارة السابقة يتم قراءتها كآلاتي : « أضف واحداً إلى قيمة المتغير days وخصص الناتج للمتغير days كقيمة جديدة ا ) .

٢ — أخذ « القراءة التالية » بموجب العارة read كآلاتي :

**read(nextreading);**

٣ — جمع « القراءة التالية » (بعد قراءتها بالطبع) إلى مجموع القراءات السابقة total بالعارة الآتية :

**total:= total + nextreading;**

٤ — اختبار شرط التوقف كالآتي :

**until total < threshold;**

وبالخروج من حلقة التكرار والاختبار يتم طبع النتائج المطلوبة بعبارة الطباعة .

(٤-٩-٢) عبارة التكرار **while-do** :

نمدنا لغة باسكال بعبارة أخرى للتكرار الذى يتضمن الشرط وهى العبارة **while** . والفارق الأساسى بين هذه العبارة والعبارة **repeat** التى عرضناها فى الفقرة السابقة أنه مع العبارة **repeat** يتم إجراء الاختبار على الشرط بعد تنفيذ العملية المطلوبة ، أما مع العبارة **while** فيتم اختبار الشرط أولاً .

مثال (٤ - ١٧) :

إذا أردنا بناء برنامج تجميع قراءات التجربة الذى عرضناه فى الفقرة السابقة باستخدام هذه العبارة الجديدة فإنه يصبح كالآتي :

```
while total <= threshold do
begin
    days := days + 1;
    read(nextreading); total := total + nextreading
end
```

*between between*

شكل (٤ - ٢٧)

ويتضح لنا من المثال أن العبارة تبدأ بكلمة **while** يعقها الشرط المطلوب يعقها كلمة **do** ثم تبدأ بعد ذلك عملية المعالجة وهى قراءة البيانات وتجميعها . وتنتهى الحلقة بعدم تحقق الشرط فينتقل البرنامج إلى العبارات التالية . ويمكن

التعبير عن الحلقة while بالألفاظ كالآتي : « طالما يسرى هذا الشرط التالى أجز العملية التالية وإلاّ فانقل إلى العبارات التالية » .

فما الفرق بين العبارتين while ، repeat ؟ .

إن العبارة while تؤدي نفس العمل الذى تؤديه العبارة repeat ، ولكن يتضح لنا من المثال السابق فرق جديد بينهما وهو ضرورة استخدام begin ، end لحصر العبارات المتعددة بداخل حلقة التكرار .

ولكن السؤال الذى ربما يخطر لنا الآن هو : ما هى الخاصية التى تتميز بها هذه العبارة على العبارة repeat ؟ وهذا سؤال فى محله .

فبرغم أن المبرمج حر فى اختياره للأداة التى يفضلها لكنه أمام تطبيقات معينة يجب عليه اختيار الأداة المناسبة ولنضرب . هذا المثال البسيط . لنفترض أنك اقترضت مبلغاً من المال من أحد أصدقائك على أن تقوم بتسديده . فى صورة أقساط شهرية . فإذا أردت أن تجعل الكمبيوتر يذكرك كل شهر بما تبقى عليك من الدين فإنك تبرمج هذه العملية الحسابية البسيطة بلغة باسكال كالآتي :

**dept := debt – payment**

حيث يدل المتغير debt على الدين المتبقى .

ويدل المتغير payment على القسط الشهرى المدفوع أما العملية الحسابية فهى عبارة عن طرح القسط من الدين وتخصيص الناتج للمتغير debt فى الطرف الأيسر والذى يمثل القيمة الجديدة للدين .

فإذا كان الدين يتم تسديده على ١٢ شهراً مثلاً فيمكنك أن تحدد عدد مرات تكرار العملية بحلقة تكرارية ذات عدّاد فتشاهد على الشاشة قيمة الأقساط الشهرية والديون المتبقية خلال العام . أما إذا لم تكن مدة الوفاء بالدين معروفة فإنك تُدخل إلى البرنامج قيمة القسط الشهرى كل مرة فيوافيك بالمبلغ المتبقى .

وسوف تستخدم البرنامج مرة بعد مرة ، ويتناقص الدين عند الدفع كل قسط حتى تصل إلى مرحلة تجد فيها الدين عدداً سالباً ومعناها أن الدين قد انتهى تسديده من الشهر السابق !!

ورغم أن هذا لا يتفق مع واقع الأمور تماماً ، لكن البرنامج الجيد البناء لا يجب أن يستمر في إعطاء النتائج التي بلا معنى . إذن فهذا البرنامج يحتاج لإضافة شرط ما .

ولكن أين نضع هذا الشرط ؟

لو أننا سوف نستخدم العبارة الشرطية IF فمن الطبيعي أن يتم اختبار المبلغ المتبقى من الدين قبل إجراء أية عملية حسابية والشرط المطلوب هو أن يكون الدين أكبر من القسط !

فإذا أردت طباعة بيانات سداد الدين على مدار عام أو عامين فبصرف النظر عن عدد مرات تكرار الحلقة فإن الشرط الذى يسبق العملية سوف يتولى إيقاف البرنامج في الوقت المناسب حتى لا يطبع أعداداً سالبة .

والذى يناظر هذا المنطق تماماً هو استخدام الحلقة while فهي تحتوى على الشرط سابقاً للعملية . أما إذا استخدمنا الحلقة repeat فإنها يمكن أن تؤدي إلى وجود أعداد سالبة في النتائج لأن الاختبار لا يتم إلا بعد إجراء العملية .

مثال (٤ - ١٨) :

ويمكننا الخروج من هذا المثال البسيط إلى حالة أكثر تعقيداً عندما تصبح المسألة مشاركة في تجارة ففى هذه الحالة يتم دفع نسبة من الربح علاوة على القسط الشهرى فإذا مثلنا نسبة الربح بالمتغير monthlyrate فإن البرنامج المطلوب يصبح كالآتي :

```

program completestatement(input,output);
var debt, monthlyrate, payment :real;

begin
  read(debt, monthlyrate, payment);
  while debt + debt*monthlyrate/100 >= payment do
  begin
    debt := debt + debt*monthlyrate/100 - payment;
    writeln('debt after next payment is ', debt)
  end;
  writeln;
  writeln('final payment required will be ',
    debt + debt*monthlyrate/100)
end.

```

البرنامج

المدخلات

القسط الأخير

#### شكل (٤ - ٢٨)

وهذا البرنامج يطبع الكشف المطلوب للسداد باستخدام حلقة تكرارية طالما أن الدين أكبر من أو يساوي فإذا قلَّ عن ذلك خرج البرنامج من الحلقة التكرارية لطبع القيمة المتبقية من الدين والتي تمثل آخر قسط . وهذه العبارة الأخيرة تصبح ذات أهمية عندما لا يكون مبلغ السداد ثابتاً .

#### مثال (٤ - ١٩) اختبار في الحساب :

من البرامج الهامة التي نحرص على أن تتضمنها كتب اللغات عموماً هي البرامج التعليمية أو التي يطلق عليها الآن التعليم بمعاونة الكمبيوتر Computer Assisted Learning (CAL) وكما نعلم أن هذه النوعية من البرامج بخلاف فائدتها في التعليم فهي يمكن أن تتطور لتستوعب أرقى فنون البرمجة لتصبح لعبة من ألعاب الكمبيوتر .

والمثال الذي نعرضه الآن لا يُعد سوى نواة صغيرة لبرنامج يختبر قدرة

التلميذ على أداء عمليات الجمع لكنه يمكن تطويره تدريجياً ليصل إلى أى درجة من التعقيد .

```

program arithmetictest(input,output);
const a = 16;  b = 25;
var nooftries, answer :integer;

begin
  writeln('lets test your arithmetic. ');
  writeln('type the answer to the following sum. ');
  write(a, ' + ', b, ' = ');
  read(answer);  nooftries :=1;

  while answer <> a+b do
  begin
    writeln;  writeln('wrong - try again. ');
    write(a, ' + ', b, ' = ');
    read(answer);  nooftries := nooftries + 1
  end;

  writeln;
  if nooftries = 1 then
    writeln('very good, got it in one!')
  else writeln('got it at last!');
  writeln('bye for now!')
end.

```

البرنامج  
اختبار في الحساب

شكل (٤ - ٢٩)

عند تشغيل البرنامج سوف يعرض عليك مسألة جمع هي :

**16 + 25**

وعليك بكتابة الإجابة باستخدام لوحة الأزرار فإذا كانت الإجابة صحيحة  
جاءتك رسالة تهنئة حارة :

**very good, got it in one !**

وإذا كانت الإجابة خاطئة جاءت الرسالة الآتية لتمنحك فرصة محاولة أخرى :

**wrong – try again**

فإذا كانت المحاولة الثانية ناجحة جاءت التهئة أقل حرارة من ذى قبل :

**got it at last !**

**bye for now !**

والغرض من هذا البرنامج هو عرض كيفية استخدام حلقة التكرار المشروطة في تكرار المحاولات عند استقبال إجابة خاطئة ، واستخدام العبارة الشرطية لطبع الرسائل المناسبة في كل حالة . وأولى المطالب اللازمة لتطوير البرنامج هي تغيير الأعداد التي يتم جمعها كل مرة ومن الأفضل أن تكون أعداداً عشوائية تتغير مع كل تنفيذ للبرنامج .

(٣ — ٩ — ٤) مقارنة بين الحلقتين **while** ، **repeat** :

الحلقة <b>repeat</b>	الحلقة <b>while</b>
تنفذ العبارات بداخل الحلقة مرة واحدة على الأقل .	من الجائز ألا تنفذ العبارات بداخل الحلقة على الإطلاق .
عندما يتحقق شرط الحلقة (true) يتوقف الكومبيوتر عن التكرار .	طالما كان شرط الحلقة نافذاً (true) يستمر الكومبيوتر في التكرار .
ما بين الكلمتين <b>repeat</b> ، <b>until</b> يمكن استخدام أى عدد من العبارات .	إذا احتوت حلقة التكرار على أكثر من عبارة لزم استخدام <b>begin</b> في البداية و <b>end</b> في النهاية .



#### (٤ - ١٠) طرق مختلفة لمعالجة البيانات بالحلقات التكرارية :

عندما يكون عدد البيانات المطلوب إدخالها ومعالجتها معروفاً مسبقاً فإن الحلقة for ذات العداد تفي بالغرض . حيث يُضبط العداد على القيمة المطلوبة ويتم إدخال البيانات ومعالجتها بالتتابع حتى يصل العداد إلى قيمته النهائية فيتوقف البرنامج .

لكن هذا ليس هو الحال دائماً . فمع الكثير من أنواع البيانات لا يكون العدد معروفاً مبدئياً . في مثل هذه الأحوال نستخدم الحلقة repeat أو الحلقة while ونصطنع شرطاً معيناً لإنهاء الحلقة . فإذا كان البرنامج يستقبل أعداداً موجبة فلا بأس من استخدام عدد سالب عندما نريد لإنهاء الحلقة التكرارية على أن يتضمن البرنامج الشرط الخاص بإنهاء الحلقة عند وجود عدد سالب .

#### مثال (٤ - ٢٠) أخطاء شهيرة :

يقوم هذا البرنامج بقراءة وجمع مجموعة من الأرقام الحقيقية الموجبة التي ندخلها إليها واحداً تلو الآخر ، فإذا أدخلنا إليه عدداً سالباً توقف البرنامج عن القراءة والجمع وطبع النتيجة .

```
program add(input,output);
var sum, next :real;
begin
  sum := 0;
  read(next);
  repeat
    sum := sum + next;
    read(next)
  until next < 0;
  writeln('sum is ', sum)
end.
```

#### شكل (٤ - ٣٠)

ونلاحظ في هذا المثال أن القيمة التالية للعدد المُدخل يتم اختبارها دائماً قبل معالجتها (جمعها على ما سبق من قيم) ، ولذلك لزم قراءة القيمة الأولى قبل الدخول في الحلقة التكرارية مما استلزم استخدام العبارة read مرتين ، مرة قبل دخول الحلقة ومرة بداخل الحلقة .

ولعله من الأفضل لاستيعاب هذه الملاحظة أن نعرض بعض الأخطاء الشهيرة التي يمكن أن يقع فيها المبتدئون عند إنشاء مثل هذا البرنامج :

**الخطأ الأول :**

```
sum := 0;
repeat
  read(next);
  sum := sum + next
until next < 0
```

**شكل (٤ - ٣١)**

يختلف هذا البرنامج عن السابق في أن عملية المعالجة (الجمع) تسبق عملية اختبار الشرط مما ينتج عنه إضافة القيمة السالبة إلى المجموع الكلي قبل توقف البرنامج .

**الخطأ الثاني :**

```
sum := 0;
read(next);
repeat
  read(next);
  sum := sum + next
until next < 0
```

**شكل (٤ - ٣٢)**

في هذا البرنامج لن يتم معالجة أول عدد مُدخل حيث أن العبارة read المخصصة لقراءة العدد الأول تلتها عبارة read التي تقع بداخل الحلقة . كما أن القيمة السالبة سوف تضاف إلى المجموع أيضاً .

### الخطأ الثالث :

```
sum := 0;
repeat
  read(next);
  sum := sum + next;
  read(next)
until next > 0
```

### شكل (٤ - ٣٣)

في هذا البرنامج سوف يتم جمع القيم الأولى والثالثة والخامسة .. إلى آخره ، وسوف يتم اختبار القيم الثانية والرابعة والسادسة ... إلى آخره ! .

### مثال (٤ - ٢١) معالجة الانتخابات بالكمبيوتر :

ربما لن تخرج عملية جمع أصوات الناخبين عن العملية السابقة لإضافة مجموعة من الأعداد المدخلة إلى الكمبيوتر بالتتابع ، فيما عدا أن الأعداد التي نقوم بجمعها هذه المرة تنتمي عادة إلى أكثر من فئة حيث تمثل كل فئة حزباً معنياً من الأحزاب المرشحة .

ولنبداً بفرز نتائج انتخابات حزبيين فقط ، ولنفترض أن مهمة البرنامج هي تجميع الأصوات التي حصل عليها كل حزب في اللجان الانتخابية المختلفة . في هذه الحالة سوف يكون لدينا فئتان منفصلتان من الأعداد ، حيث يتم جمع عناصر كل فئة على حدة ، كما يجب أيضاً في هذه الحالة استخدام نهاية مستقلة لكل فئة من البيانات ، بمعنى إدخال عددين سالبين عند انتهاء البيانات .

وهذا هو البرنامج :

```

program election(input,output);
var party1next, party2next,
    party1overall, party2overall :integer;

begin
    party1overall := 0; party2overall := 0;
    read(party1next, party2next);
    repeat
        party1overall := party1overall + party1next;
        party2overall := party2overall + party2next;
        read(party1next, party2next)
    until party1next < 0;

    writeln('party1: ', party1overall, '
            'party2: ', party2overall)
end.

```

البرنامج

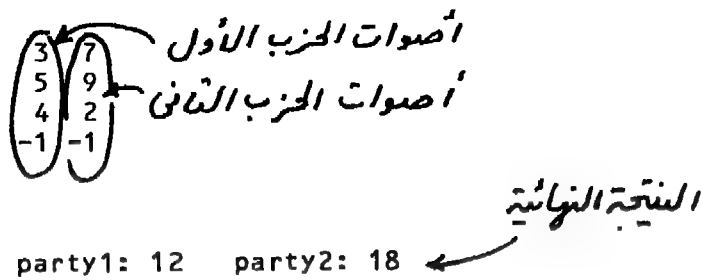
معالجة الانتخابات

### شكل (٤ - ٣٤)

في هذا البرنامج تم استخدام المتغيرات الآتية :

- |                 |                                       |
|-----------------|---------------------------------------|
| ● party1next    | مجموع أصوات الحزب الأول في أى لجنة .  |
| ● party2next    | مجموع أصوات الحزب الثانى في أى لجنة . |
| ● party1overall | مجموع أصوات الحزب الأول الكلية .      |
| ● party2overall | مجموع أصوات الحزب الثانى الكلية .     |

وعند تشغيل البرنامج فإنه ينتظر إدخال قيمتين لكل من أصوات الحزبين في لجنة ما ، وتفصل القيمتان بمسافة خالية ، وعند الضغط على الزر ENTER (أو RETURN) ينتظر إدخال قيمتين جديدتين وهكذا حتى ندخل له عددين سالبين فيتوقف عن الجمع ويطبع النتيجة الممثلة لمجموع أصوات كل حزب على حدة . وهذا مثال لتنفيذ البرنامج :



### شكل (٤ - ٣٥)

ولعلنا نلاحظ في هذا البرنامج أنه بالرغم من إدخال عددين سالبين لإنهاء العملية التكرارية لكن الاختبار يتم على عدد واحد منهما فقط ذلك هو العدد الأول المناظر للمتغير party1next . لكنه من الضروري أن ندخل عددين سالبين لأن تصميم البرنامج يستدعي ذلك بموجب العبارة :

**until party1next<0;**

فإذا رغبتنا في أن ندخل قيمة سالبة واحدة كنهاية للبيانات لزم ذلك إجراء تغيير طفيف في تصميم البرنامج حيث يتم قراءة أصوات الحزب الثاني في بداية الحلقة وقراءة أصوات الحزب الأول قبل الشرط مباشرة وفي هذه الحالة يمكن إجراء الاختبار على متغير أصوات الحزب الأول فقط كما في الشكل (٤ - ٣٦) .

```

read(party1next);
repeat
    read(party2next);
    party1overall := party1overall + party1next;
    party2overall := party2overall + party2next;
    read(party1next)
until party1next < 0

```

### شكل (٤ - ٣٦)

وهناك طريقة أخرى لتصميم البرنامج بحيث يتم إدخال بيانات كل حزب على حدة يعقبها العدد السالب الذى ينهى إدخال البيانات .

وهذه الطريقة أكثر مرونة لا سيما إذا كان أحد الحزبين غير ممثل فى أحد الدوائر وستترك هذه المعالجة كتمرين على أن نقدم الحل فى نهاية الكتاب .

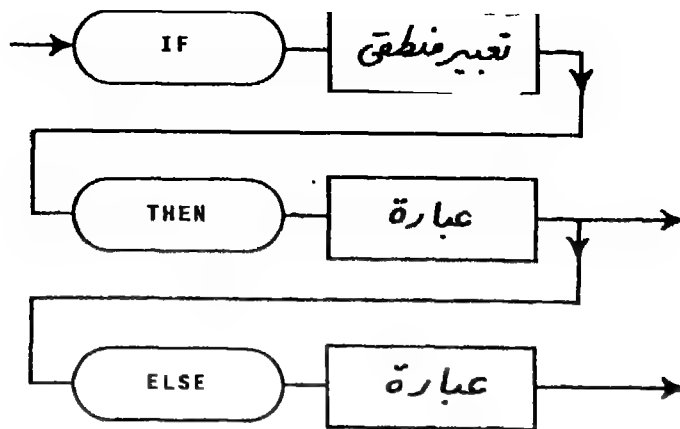


## ملخص :

نلخص فيما يلي القواعد التي وردت في هذا الباب بالرسم ولا يفوتنا تذكر المصطلحات الآتية التي عرضناها في الأبواب السابقة :

statement	عبارة
expression	تعبير
boolean expression	تعبير منطقي (أو بولياني)
identifier	اسم بيان
variable identifier	متغير
constant	ثابت
integer	صحيح
real	حقيقي
unsigned integer	عدد صحيح بدون إشارة
label	عنوان

ونلاحظ في الرسومات التالية أن الكلمات المكتوبة بالحروف الكبيرة هي كلمات من لغة باسكال مثل DO , FOR , .... أما الكلمات المكتوبة باللغة العربية فهي كلمات ذات معنى (كالمصطلحات السابق ذكرها) وهي تستبدل بما يناسبها من بيانات أو تعبيرات أو متغيرات ... إلخ .



شكل (٤ - ٣٧)  
العبارة الشرطية (IF)

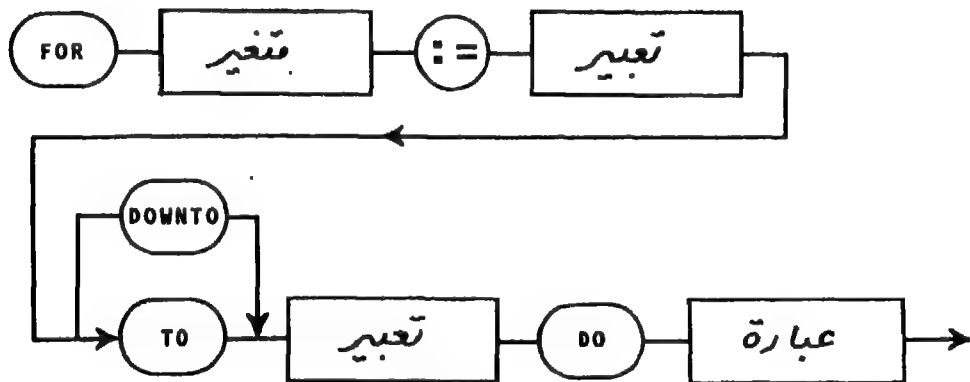
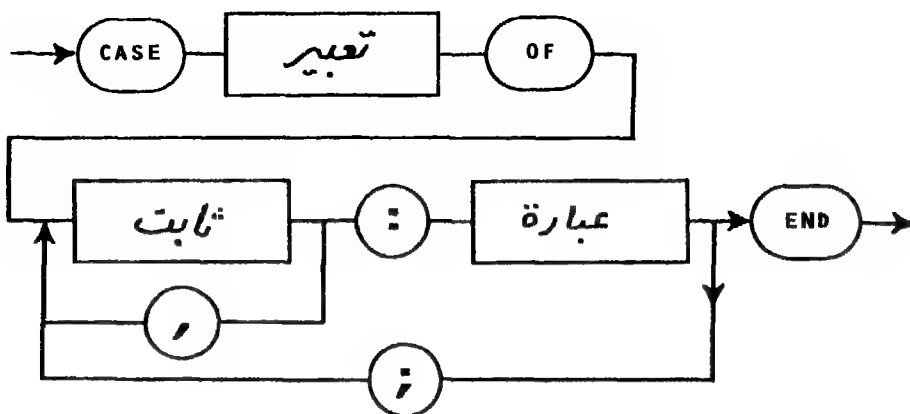


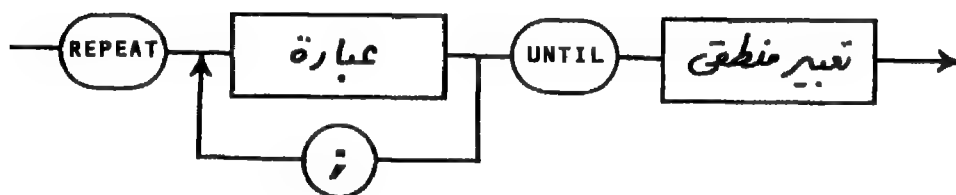
Fig. 6.2

شكل (٤ - ٣٨)  
عبارة التكرار (for)





شكل (٤ - ٣٩)  
عبارة الحالات المتعددة (case)



شكل (٤ - ٤٠)  
عبارة التكرار المشروط (repeat)



شكل (٤ - ٤١)  
عبارة التكرار المشروط (while)



## ■ تمارين الباب الرابع :

س (٤ - ١) اكتب برنامجاً يطبع جدول الضرب للأعداد المختلفة كالمثال الآتي :

```
5
1×5=5
2×5=10
3×5=15
4×5=20
5×5=25
6×5=30
7×5=35
8×5=40
9×5=45
10×5=50
```

## شكل (٤ - ٤٢)

س (٤ - ٢) ينتج عن استخدام العبارة :

**writeln('\*:i)**

طبع نجمة واحدة متبوعة بعدد (1-i) مسافة خالية . والواقع أنه يجوز أن يستبدل المتغير i بأي تعبير صحيح (integer) إذا دعت الحاجة .

اكتب برنامجاً يقرأ قيمة متغير صحيح n ويطبع الرقم 7 باستخدام النجوم في عدد n من السطور . وعلى سبيل المثال فلو كان n=6 فإن الخرج يصبح كالتالي :

```

*****
 *
  *
   *
    *
     *
      *

```

شكل (٤ - ٤٣)

س (٤ - ٣) اكتب برنامجاً بلغة باسكال لرسم مثلث يحتمل عدد  $n$  من السطور حيث  $n$  هو متغير صحيح يقوم البرنامج بقراءته عند التنفيذ . وعلى سبيل المثال إذا كانت  $n=7$  فإن الحرج يكون في شكل (٤ - ٤٤) .

```

      *
     * *
    * * *
   * * * *
  * * * * *
 * * * * *
* * * * *

```

شكل (٤ - ٤٤)

س (٤ — ٤) المطلوب إعداد برنامج لحل بيع للأحذية يستخدم المقاييس البريطانية ، لكي يرشد البائع إلى مقاس الحذاء بالنظام الأمريكي إذا اقتضى الأمر .

والعلاقة بين النظامين ليست علاقة خطية كما أنها تختلف بحسب ما إذا كان الحذاء « حريمي » أو « رجالي » .

وهذا نموذج للخروج المطبوع من البرنامج المطلوب :

<u>Men's shoes</u>		الأحذية الرجالي				
British	7	8	9	10	11	
American	7.5	8.5	9.5	10.5	11.5	

<u>Women's shoes</u>		الأحذية الحريمي				
British	3	4	5	6	7	
American	4.5	5.5	6.5	7.5	8.5	

### شكل (٤ — ٤٥)

أما المدخلات للبرنامج فهي مقاس الحذاء بالنظام البريطاني وأحد الرقمين 1 أو 0 لتمثيل الحذاء الرجالي أو الحريمي .

يمكنك إضافة المقاييس الفرنسية أيضاً لهذا البرنامج .

س (٤ — ٥) اكتب برنامجاً يقرأ العدد الممثل للشهر مثل jan, feb, mar,... ويطبع عدد أيام هذا الشهر ، وذلك بفرض أن السنة غير كبيسة .

س (٤ — ٦) بتطوير البرنامج السابق يمكن أن يقوم البرنامج أيضاً بكتابة اسم الشهر مستنداً عليه برقمه كالمثال الآتي :

please enter month number

12

december.. this month contains: 31 days

>

### شكل (٤ - ٤٦)

س (٤ - ٧) بتطوير البرنامج السابق اجعل البرنامج يقرأ السنة علاوة على شهر ويقوم باختبار السنة إذا ما كانت كبيسة أم لا ثم يطبع عدد أيام الشهر .

(ملاحظة : فبراير في السنة الكبيسة ٢٩ يوماً  
فبراير في السنة غير الكبيسة ٢٨) .

س (٤ - ٨) في أحد معاهد الكومبيوتر يتم ترقيم المناهج التي تدرس بالمعهد بالأرقام من 1 إلى 10 ، ويتم أداء المحاضرات في فترات قدرها ساعتين لكل محاضرة .

والجدول الآتي بعد يمثل المواعيد المختلفة لهذه المناهج والمحاضرات حيث يوضح الجدول رقم المنهج (course'1,2,...) واليوم (. ,thu, mon) والساعة (9am, 10am...)

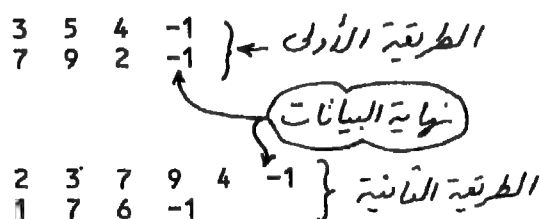
Course 1,2	thu 9am	fri 10am
Course 3	mon 10am	thu 10am
Course 4,5	mon 11am	tue 11am
Course 6,7	tue 9am	wed 2pm
Course 8	mon 12am	thu 9am
Course 9	tue 10am	wed 11am
Course 10	fri 9am	fri 11am

### شكل (٤ - ٤٧)

اكتب برنامجاً يمكن استخدامه للاستعلام عن مواعيد المحاضرات حيث يقرأ البرنامج رقم المنهج ويطبع رسالة بالمواعيد المحددة للمحاضرة (اليوم والساعة) خلال الأسبوع .

س (٤ — ٩) في المثال السابق إذا عبرنا عن أيام الأسبوع بالأرقام من 1 إلى 6 ، وساعات اليوم بالأرقام من 1 إلى 6 (بدءاً من الساعة ٩ صباحاً (9am) وحتى الثانية بعد الظهر (2pm) . فإنه يمكن تمثيل الفترة الزمنية للمحاضرة بعدد واحد ذي رقمين الرقم الأول يعبر عن اليوم والرقم الثاني يعبر عن الساعة . معنى ذلك أن العدد 12 يعنى يوم السبت (1) الساعة العاشرة صباحاً (2) . اكتب برنامجاً يستقبل هذا الكود الممثل لليوم والساعة ويطبع على الشاشة أية محاضرات تعقد في هذا الموعد .

س (٤ — ١٠) اكتب برنامجاً لجمع الأصوات التي يحصل عليها حزبان في الدوائر المختلفة على أن يتم تشغيل البرنامج بأحد الطرق الآتية :



شكل (٤ — ٤٨)

وكما نرى فإنه مع الطريقة الثانية فإن الحزب الثاني غير ممثل في بعض الدوائر الانتخابية لذلك فقد انتهت بياناته قبل الآخر .





## الباب الخامس

# منشآت التحكم (CONTROL STRUCTURES)



## مفتح

إذا شرعت في برمجة مشكلة ذات أبعاد حقيقية فإنك سوف تحتاج لبعض المهارات في إنشاء البرنامج خاصة عندما تتشابه العلاقات التي تربط بين المتغيرات في الأجزاء المختلفة للبرنامج . فقد يتطلب الأمر في بعض الأحيان أن تحتوي حلقة تكرارية على عبارة شرطية بداخلها أو تحتوي العبارة الشرطية على حلقة تكرارية أو أن تتداخل الحلقات التكرارية بداخل بعضها البعض .

هذه التراكيب المختلفة تسمى منشآت التحكم وهي وإن كانت تتطلب مهارات معينة في المعالجة لكنها مع ذلك خبرات معروفة يمكن اكتسابها من خلال الأمثلة التي نعرضها في هذا الباب والتي تعالج مشكلات حقيقية .

## (٥ - ١) الحلقات التكرارية المحتوية على عبارات شرطية :

علمنا من قبل أن الحلقة التكرارية for تتبع الصيغة الآتية :

**for variable: = expression to expression do statement**

والعبارة المطلوب تكرارها (statement) قد تكون على أية صورة من صور عبارات باسكال التي درسناها . وإحدى الصور الممكنة هي العبارة الشرطية . ولنر مثلاً لهذه الحالة :

مثال (٥ - ١) معالجة رياضية :

إذا أعطيت عدداً صحيحاً مثل 18 فإنه يمكن إيجاد الأعداد التي يقبل القسمة عليها بدون باق والتي تنحصر بين 2 ، 9 بطريقة يدوية ، وهي الأعداد :

2 , 3 , 6 , 9

فإذا أردنا أن نوكل هذا العمل إلى الكمبيوتر فإن دالة الباقي MOD التي نعرفها بها من قبل يمكن أن تساعدنا في برجة الحل ، حتى يكون أكثر عمومية . فلو عبرنا عن العدد المقسوم (18) بالمتغير giveninteger ولو عبرنا عن الأعداد المقسوم عليها بالمتغير i فإنه لجميع القيم المطلوبة (مثل 2 ، 3 ، 6 ، 9) يكون :

**giveninteger MOD i=0**

هذا هو شرط القسمة بدون باق .

والبرنامج التالي يقرأ عدداً صحيحاً ويستقبله في المتغير giveninteger ثم يستخدم الحلقة التكرارية (من 2 إلى 9) لإيجاد الأعداد الصحيحة الموجبة التي تقع في هذا النطاق والتي يقبل القسمة عليها المتغير giveninteger بدون باق .

والعبارة الشرطية هنا تقع بداخل حلقة التكرار حيث أن الشرط يتم اختباره لكل عدد من الأعداد بين 2 ، 9 .

```
program factors(input, output);
var giveninteger, i : integer;
begin
  read(giveninteger);
```

البرنامج

```
  for i := 2 to 9 do
```

```
    if giveninteger mod i = 0 then
      writeln(i, ' divides into the given integer.')
```

```
end.
```

شكل (٥ - ١) أ

والآتي بعد مثال لتنفيذ البرنامج مع العدد 18 :

```
2 divides into the given integer.
3 divides into the given integer.
6 divides into the given integer.
9 divides into the given integer.
```

التنفيذ

شكل (٥ - ١) ب

مثال (٥ - ٢) المراقبة الرقمية للإنتاج :

عالجنا في الباب الرابع في البرنامج (tolerance2) عملية مضاهاة مجموعة من

الأعداد برقم قياسي بهدف حصر الأعداد التي تقترب قيمتها من قيمة الرقم القياسي .

وقد استخدمنا في هذا البرنامج العبارة الشرطية بداخل حلقة التكرار .  
ولعلنا في هذا المثال نضيف لمسة واقعية إلى المعالجة المذكورة لتوضيح فائدة مثل هذا البرنامج . ففي مصانع الإنتاج بالجملة عادة ما يستخدم الكمبيوتر في مراقبة الإنتاج . فإذا كان المصنع يقوم بإنتاج قضبان معدنية ذات أطوال معينة . فإن جهاز القياس الذي يقوم باختبار طول القضبان الخارجة من خط الإنتاج عادة ما يكون مبرمجاً (كمبيوتر ذو غرض خاص) .

وفي المثال الذي نحن بصدده سوف نجرى عملية محاكاة simulation للعملية الصناعية المذكورة . فالأعداد التي نستقبلها في البرنامج تمثل القياسات الناتجة من جهاز قياس الأطوال . ويقوم البرنامج بحصر الأطوال التي لا تختلف عن الطول القياسي إلا في حدود العُشر . ويتوقف عمل البرنامج عندما يستقبل عدداً سالباً .

وفي هذا البرنامج سوف نستخدم المتغيرات التالية :

- standard ثابت مسمى قيمته 6.37 للتعبير عن طول قياسي معين .
  - length متغير حقيقي يعبر عن الطول .
  - nowithin متغير صحيح لحصر عدد الأطوال التي تقع في النطاق المقبول .
  - nowithout متغير صحيح لحصر عدد الأطوال التي تقع خارج النطاق المقبول .
- ويتمى البرنامج بطبع قيمتى المتغيرين الأخيرين .

```

program tolerance3(input,output);

const standard = 6.37;
var length : real;
    nowithin, nowwithout : integer;

begin

    nowithin := 0;
    nowwithout:= 0;
    read(length);

    repeat
        if abs(length - standard) < 0.1 then
            nowithin := nowithin + 1
        else
            nowwithout:= nowwithout + 1;

        read(length)
    until length < 0;

    writeln(nowithin, ' values are within tolerance.')
```

البرنامج

المراقبة الرقمية للإنتاج

```

    writeln(nowwithout, ' values are outside tolerance.')
```

end.

### شكل (٥ - ٢)

نتائج ستة تلاميذ في ست مواد مختلفة في الامتحان الشهري ممثلة في صورة جدول (مصفوفة) .

### (٥ - ٢) الحلقات التكرارية المتداخلة *nested loops* :

تحتاج بعض التطبيقات إلى استخدام أكثر من حلقة تكرارية « تحتضن » بعضها البعض . وأهم هذه التطبيقات هو الجداول ، مثل الجدول الذي يحتوي

على نتيجة الطلبة في أحد الكليات في نهاية العام الدراسي .

ولو أننا أردنا تمثيل درجات طالب واحد في ثمانى مواد دراسية على سبيل المثال ، فإن برنامج القراءة سوف يكرر عملية القراءة ثمانى مرات . فإذا كان البرنامج يقرأ درجات عشرة طلبة فإننا نحتاج لحلقتين تكراريتين الأولى تتغير مع كل طالب (من واحد إلى عشرة) والثانية تتغير مع كل مادة (من واحد إلى ثمانية) لكل طالب من الطلبة العشرة . والجدول الموضح في الشكل (٥ - ٣) يبين درجات بعض التلاميذ في صورة جدول . ونرى أن درجات التلميذ الواحد تحتل صفّاً كاملاً أى أن الصف يعبر عن عدد المواد الدراسية بينما يعبر العمود عن عدد الطلبة .

ومن سبق له دراسة لغة ييسك فإنه لا بد قد صادف مصطلح المصفوفة (matrix) للتعبير عن هذا الجدول .

والمثال الآتى يوضح كيفية قراءة درجات خمس وعشرين تلميذاً في خمس مواد دراسية وجمع درجات كل تلميذ على حدة لبيان نتيجته ومجموعه . وقد استخدمنا في هذا البرنامج حلقة خارجية ذات عدّاد (من 1 إلى 25) تمثل الطلبة وحلقة داخلية ذات عدّاد من (1 إلى 5) تمثل المواد أى أن الحلقة الداخلية تتغير خمس مرات مع كل قراءة من قراءات عدّاد الحلقة الخارجية أى أنها تنفذ  $25 \times 5$  مرة !



الاسم	العلوم والصحة	الهندسة	الحساب	والتاريخ	الجغرافيا	التربية البدنية	اللغة العربية	المواد الاسماء
محمد سمير	13	20	18	15	17	8		
سوان فاروق	14	17	17	12	15	12		
عشام مازور	20	19	20	18	20	10		
حازم امامة	19	20	20	18	19	13		
كريم راجد	20	19	20	19	18	12		
ياني الحسين	17	17	15	19	17	15		

نتائج ستة تلاميذ في ست مواد مختلفة في الامتحان الشهري ممثلة في صورة  
جدول ( مصفوفة )

شكل ( ٥ - ٣ )

مثال ( ٥ - ٣ ) نتيجة فصل دراسي :

المتغيرات المستخدمة في البرنامج هي :

**candidate** متغير العداد للحلقة الخارجية (حلقة عدد الطلبة) وبالطبع

يمكن استخدام أى حرف مثل x أو y للتعبير عن المتغير

ولكن كلمة candidate تذكر دائماً بأن هذه هي

حلقة عدد الطلبة عندما تقرأ البرنامج . وهذه خاصية مفيدة من خصائص لغة باسكال حيث تسمح باستخدام المتغيرات ذات الأسماء الطويلة كمتغيرات للعدادات .

**exam** متغير العداد للحلقة الداخلية (حلقة عدد المواد)

**total** متغير مجموع الدرجات . ولأنه يستخدم كوعاء لتجميع درجات كل طالب فإنه يتم تفريغه (مساواته بالصفر) عند بداية كل دورة من دورات الحلقة الخارجية .

**mark** متغير يمثل درجة أى مادة من المواد .

**average** متغير يمثل متوسط مجموع الدرجات وهو يساوى مجموع الدرجات مقسوماً على عدد المواد .

ويعتبر الطالب ناجحاً إذا حصل على متوسط قدره 50 أو أكثر .

وبالطبع يمكن تطوير هذا البرنامج باستقبال أسماء التلاميذ أيضاً وطبعها في المخرج حتى تكون النتيجة مقبولة شكلاً . فالبرنامج في صورته الحالية يطبع أرقام التلاميذ بدلاً من أسمائهم .

```
program examarks(input,output);
```

البرنامج

```
var candidate, exam, total, mark : integer;
    average : real;
begin
```

```
  for candidate := 1 to 25 do
  begin
```

الحلقة الخارجية

```
    total := 0;
    for exam := 1 to 5 do
    begin
      read(mark);
      total := total + mark;
    end;
```

الحلقة الداخلية

نتيجة امتحان

```
    average := total/5;
    if average >= 50 then
      writeln('candidate ', candidate,
        ' passed, average is ', average)
    else
      writeln('candidate ', candidate,
        ' failed, average is ', average)
```

شرط النجاح

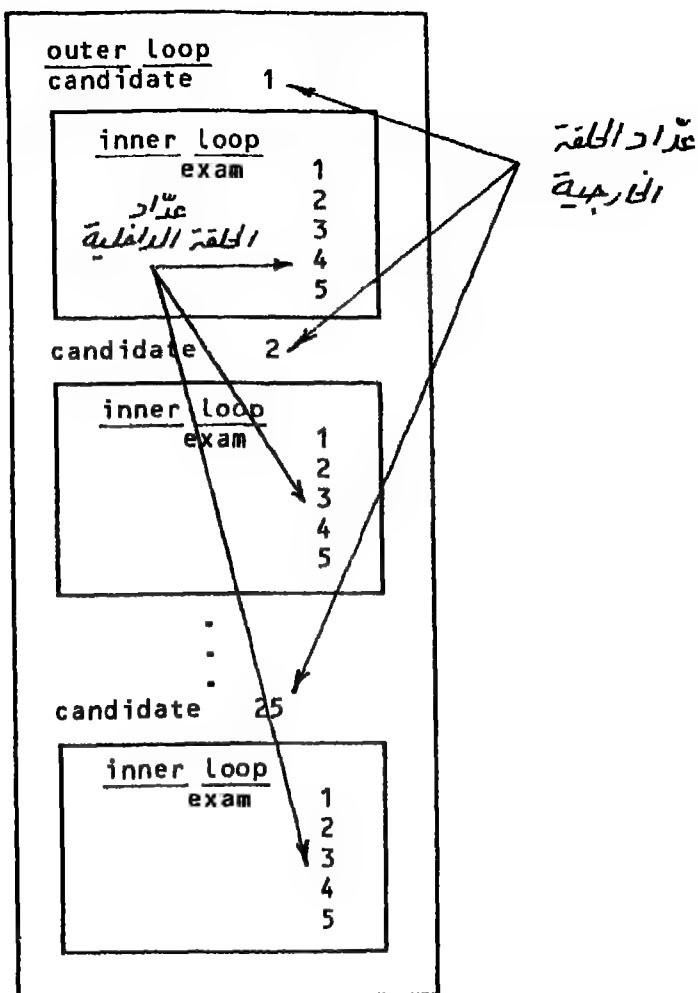
```
  end
```

شرط الرسوب

```
end.
```

### شكل (٥ - ٤)

والشكل (٥ - ٥) يوضح مجرى الأحداث في البرنامج . فالحلقة الخارجية تبدأ بالرقم 1 الذي يمثل الطالب الأول (candidate=1) . ويظل العداد محتفظاً بالرقم 1 أثناء تنفيذ الحلقة الداخلية بأكملها والتي تبدأ من (exam=1) وتتكرر خمس مرات حتى تصل إلى (exam=5) ممثلة عدد الامتحانات أو عدد المواد الدراسية . وبعد طبع نتيجة الطالب الأول تتغير قيمة العداد للحلقة الخارجية لتصبح 2 ممثلة الطالب الثاني وهكذا حتى الطالب الخامس والعشرين .



شكل (٥ - ٥)  
كيف تجري الأحداث في البرنامج

مثال (٥ - ٤) الرسم بالحلقات التكرارية المتداخلة :

قدمنا من قبل بعض أمثلة وتدريبات على الرسم لكن استخدام الحلقات

التكرارية المتداخلة يمكننا من رسم أشكال أكثر تعقيداً كالرسم الموضح  
شكل (٥ - ٦) .

```

.....
.....*.....
.....****.....
.....*****.....
.....*****.....
.....*****.....
.....*****.....
.....

```

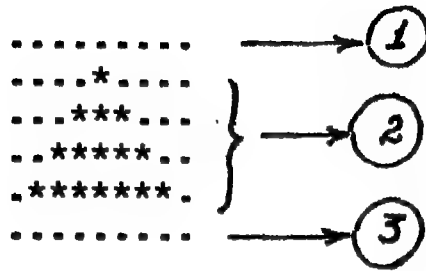
### شكل (٥ - ٦)

كيف يتم بناء هذا الرسم كمنطق في البرنامج ؟

يمكننا تمثيل المنطق الذى نرسم به مثل هذا الشكل بالخطوات المنطقية الآتية  
التي يطلق عليها الكود الكاذب (pseudo code) والتي تكافئ خريطة تسلسل  
المنطق (flowchart) :

- اقرأ عدد الصفوف في الشكل المراد رسمه .
- احسب عرض الشكل :
- ١ - كرر الآتى :
- ارسم نقطة
- حتى تصل إلى آخر عرض الشكل
- اترك سطرًا خالياً
- ٢ - كرر الآتى بدءاً من الصف رقم 1 :
- ارسم صفاً من صفوف المثلث بالعدد المناسب من النقاط على  
الجانبين حتى تصل إلى نهاية عدد الصفوف .
- ٣ - كرر الآتى :
- ارسم نقطة
- حتى تصل إلى آخر عرض الشكل .

والروتينات ١ ، ٢ ، ٣ المشار إليها تمثل خطوات رسم الأجزاء التي تحمل نفس الأرقام في الشكل (٥ - ٧) :



شكل (٥ - ٧)

أما عرض الشكل فيمكن حسابه بمعلومية عدد الصفوف على أساس العلاقة التالية :

$$\text{عرض الشكل} = \text{عدد الصفوف} * ٢ + ١$$

أما الجزء الذي يحتاج للتفصيل فهو رسم صفوف المثلث (الجزء رقم ٢) من الشكل) ويمكن وصف ذلك بالمنطق الآتي :

- احسب عدد النجوم في الصف وليكن (noofstars)
- احسب عدد النقط على جانبي النجوم في الصف (noofdots)
- كرّر الآتي :
- ارسم نقطة
- حتى تبلغ قراءة العداد القيمة noofdots
- كرّر الآتي :
- ارسم نجمة
- حتى تبلغ قراءة العداد القيمة noofstars
- كرّر الآتي :
- ارسم نقطة
- حتى تبلغ قراءة العداد القيمة noofdots

ويمكن التعبير عن عدد النجوم في الصف المعين بالعلاقة :

$$\text{عدد النجوم} = ٢ * \text{رقم الصف} - ١$$

\* \* (لاحظ أن الصف الأول هنا هو السطر الثاني في الشكل كله حيث أن الصف الأول هنا معناه الصف الأول من الشكل المثلثي المرسوم بالنجوم) .

$$\text{عدد النجوم في الصف الأول : } ٢ * ١ - ١ = ١$$

$$\text{النجوم في الصف الثاني : } ٢ * ٢ - ١ = ٣$$

$$\text{النجوم في الصف الثالث : } ٢ * ٣ - ١ = ٥$$

وهكذا ..

أما عدد النقط في كل سطر فيتم التعبير عنها بالعلاقة :

$$\text{عدد النقط على كل جانب} = \text{عدد الصفوف} + ١ - \text{رقم الصف}$$

ولاختبار هذه العلاقة نضرب الأمثلة الآتية :

عدد نقط الصف الأول =  $4 = 1 + 1 - 1 = 1$

عدد نقط الصف الثاني =  $4 = 1 + 1 - 2 = 3$

عدد نقط الصف الثالث =  $4 = 1 + 1 - 3 = 2$

وهكذا ...

وهذا المنطق يجب أن يندمج في منطق البرنامج الكلي ضمن الروتين الثاني لرسم مثلث النجوم . والشكل رقم (٥ - ٨) يوضح البرنامج الكامل لرسم الشكل المطلوب وقد استخدمت فيه المتغيرات الآتية :

- noofrows عدد الصفوف (للمثلث فقط)
- widthofpicture عرض الشكل ( مقاساً بالبنة )
- ch متغير العداد
- rowno رقم الصف
- noofstars عدد النجوم
- noofdots عدد النقاط

والبرنامج نفس المنطق الذى سبق عرضه مع ملاحظة إضافة بداية ونهاية للروتين رقم (٢) لاحتوائه على عدد كبير من الخطوات وقد تم وضع الروتين كله داخل بروتاز للإيضاح .



```
program framedtriangle(input,output);
```

البرنامج

```
var noofrows, widthofpicture, ch,  
    rowno, noofstars, noofdots : integer;
```

```
begin
```

```
  read(noofrows);
```

رسم السطر الأول

```
  widthofpicture := 2*noofrows + 1;
```

```
  for ch := 1 to widthofpicture do write(' ');
```

```
  writeln;
```

```
  for rowno := 1 to noofrows do  
  begin
```

```
    noofstars := 2*rowno - 1;
```

```
    noofdots := noofrows + 1 - rowno;
```

```
    for ch := 1 to noofdots do write(' ');
```

```
    for ch := 1 to noofstars do write('*');
```

```
    for ch := 1 to noofdots do write(' ');
```

```
    writeln
```

```
  end;
```

```
  for ch := 1 to widthofpicture do write(' ');
```

```
  writeln
```

```
end.
```

رسم السطر الأخير

## شكل (٥ - ٨)

وعند تنفيذ البرنامج يبدأ باستقبال عدد صفوف المثلث المطلوب رسمه وعلى أساس ذلك يتحدد حجم الشكل كله جرب البرنامج باستخدام أعداداً مختلفة وشاهد النتائج .

## (٥ - ٣) العبارات الشرطية المتداخلة *nested if-statements* :

علمنا من قبل أن العبارات الشرطية يجوز أن تقع بداخل حلقة تكرارية منعاً لتكرار كتابة العبارة الشرطية عدة مرات بداخل البرنامج ، ولكن هذه الحالة تنطبق فقط على العبارة الشرطية المتكررة .

كما تعاملنا من قبل مع العبارة الشرطية الكاملة (if-then-else ذات النتيجة

الأصلية والنتيجة البديلة .

ولكن هناك حالات تتعقد فيها الشروط وتتشابك بحيث يضطر المبرمج أن يكتب عدة عبارات شرطية متتابعة لوصف المشكلة كالمثال الآتي :

مثال ( ٥ - ٥ ) تمثيل البدائل المتعددة :

الجدول الآتي يمثل الحالات المختلفة التي يلتقي بها مندوب شركة التأمين وعلى أساسها يحدد نوع التأمين الذي يمكنه منحه للسيارة على أساس سعة الماكينة ، وعمر السائق ، وعدد الحوادث التي وقعت للسيارة من قبل . فالسعة تحدد إذا ما كانت السيارة كبيرة أو صغيرة وبالتالي تدل على ثمنها وتحمل شركة التأمين مسؤولية أكبر من ناحية التعويض . وعمر السائق يصبح عنصراً هاماً للشركة إذا كان يقل عن ٢١ سنة ، فمن المتوقع أن يكون السائق أكثر حياءً للمخاطرة وأكثر تعرضاً للحوادث وهذا أيضاً يرفع مبلغ التأمين على السيارة . ومع ذلك فحكم السن ليس نهائياً بل يضاف إليه تاريخ الحوادث التي وقعت للسائق من قبل وثبتت إدانته فيها فإذا كانت أكثر من ثلاثة رفضت الشركة منح بوليصة التأمين نهائياً وهكذا ..

والمطلوب من البرنامج أن يطبع الجدول الموضح على الشاشة :

العمر age	سعة الماكينة engine size	عدد مخالفات الإذانة convictions	الرسالة على الشاشة message
>=21	>=2000	>=3	policy loaded by 45%
>=21	>=2000	< 3	policy loaded by 15%
>=21	< 2000	>=3	policy loaded by 30%
>=21	< 2000	< 3	no loading
< 21	>=2000	>=3	no policy to be issued
< 21	>=2000	< 3	policy loaded by 60%
< 21	< 2000	>=3	policy loaded by 50%
< 21	< 2000	< 3	policy loaded by 10%

النسبة المئوية للزيادة في القيمة  
لا تخرج البوليصة

شكل ( ٥ - ٩ )

أحد الطرق الممكنة لتمثيل هذا الجدول هو البرنامج شكل (٥ — ١٠) وهو يستخدم المتغيرات الآتية :

#### ١ — المتغيرات المنطقية (boolean) :

- over21** : يدل على الشخص الذى يزيد عمره عن أو يساوى ٢١ عاماً
- largecar** : يدل على السيارة الكبيرة التى تزيد سعتها عن ٢٠٠٠
- riskdriver** : يدل على السائق المخاطر الذى زادت مرّات إداثته عن ثلاثة .

#### ٢ — المتغيرات الصحيحة (integer) :

- age** : العمر
- cc** : سعة السيارة
- convictions** : عدد مرّات الإداثة

والبرنامج عبارة عن ترجمة حرفيّة لسياسة التأمين التى يعرضها الجدول شكل (٥ — ٩) لذلك فهو يحتوى على عبارات شرطية بعدد سطور الجدول تماماً .

```

program policy(input,output);
    البرنامج ①
    var over21 , largecar , riskdriver : boolean;
        age , cc , convictions : integer;

    begin
        read(age,cc,convictions);

        over21 := age >= 21;
        largecar:= cc >= 2000;
        riskdriver:= convictions >= 3;

        if over21 and largecar and riskdriver
        then writeln('policy loaded by 45 percent');

        if over21 and largecar and not riskdriver
        then writeln('policy loaded by 15 percent');

        if over21 and not largecar and riskdriver
        then writeln('policy loaded by 30 percent');

        if over21 and not largecar and not riskdriver
        then writeln('no loading');

        if not over21 and largecar and riskdriver
        then writeln('no policy to be issued');

        if not over21 and largecar and not riskdriver
        then writeln('policy loaded by 60 percent');

        if not over21 and not largecar and riskdriver
        then writeln('policy loaded by 50 percent');

        if not over21 and not largecar and not riskdriver
        then writeln('policy loaded by 10 percent')

    end.
    
```

شكل (٥ - ١٠)

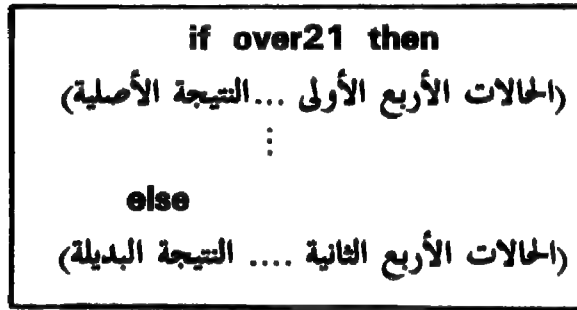
بالتأكيد هناك طريقة أفضل بل طرق أفضل من هذه الطريقة البدائية التي تستهلك وقت الكمبيوتر .

ومن أفضل الطرق التي يمكن بواسطتها إنشاء برنامج كهذا هي العبارات الشرطية المتداخلة (nested) التي تحتضن بعضها البعض . بمعنى أن نتيجة العبارات الشرطية (سواء الأصلية أو البديلة) يمكن أن تحتوى على عبارة شرطية أخرى . فإذا ألقينا نظرة ثانية على الجدول وعلى خانة « العمر » بالتحديد لوجدناه يعالج حالتين :

(١) إذا كان العمر ٢١ سنة أو أكثر .

(٢) وإذا كان العمر أقل من ٢١ سنة .

يمكن تمثيل هاتين الحالتين بعبارة شرطية واحدة كالآتي :

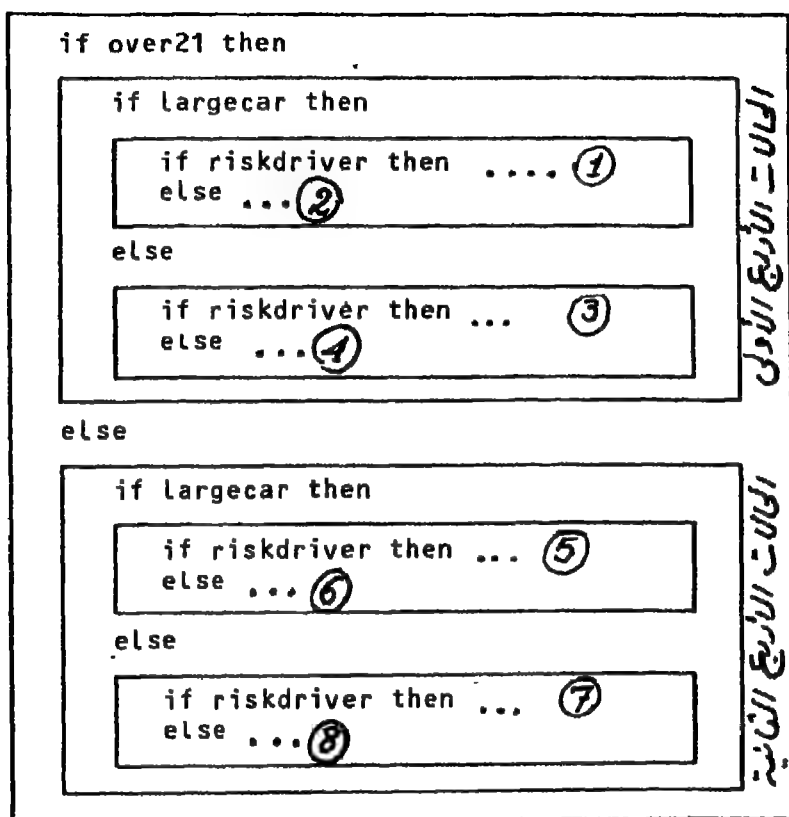


وبلى كلمة then جميع النتائج المترتبة على هذا الشرط (أكبر من ٢١ سنة) وبلى كلمة else جميع النتائج التي تترتب على هذا الشرط (أقل من ٢١ سنة) .

والحالات الأربع الأولى تتضمن شروطاً أخرى خلاف عمر السائق مثل سعة سيارته وعدد مرّات إيداعه في حوادث سيارات . لذلك فالنتيجة الأصلية التي تعقب كلمة then سوف تحتوى على عبارة شرطية أو أكثر ..

وكذلك الحال بالنسبة للنتيجة البديلة التي تعقب العبارة else . أى أن العبارة الشرطية سوف تحتضن بداخلها عبارات شرطية أخرى فتبدو كما في الشكل (٥ - ١١) ونلاحظ أن الأرقام التي بداخل الدوائر تعبّر عن

الرسالات المختلفة التى تظهر على الشاشة ممثلة الحالات الثمانية بالترتيب الوارد فى الجدول شكل (٥ - ٩) .



شكل (٥ - ١١)

إن هذه العبارات الشرطية المتداخلة توفّر وقت الكمبيوتر فلو كان طالب البوليصه يخضع للشرط الثامن مثلاً فإنه مع البرنامج شكل (٥ - ١٠) سيقوم الكمبيوتر باختبار صحة الشروط السبعة الأولى حتى يصل إلى الحالة الثامنة التى تنطبق عل طالب البوليصه .

أما مع هذا المنطق الجديد شكل (٥ - ١١) فإن الكمبيوتر سوف يخرج من الحالات الأربع الأولى بمجرد اختبار واحد فقط وهو اختبار العمر (لأن جميع هذه الحالات تمثل النتيجة الأصلية للعبارة الشرطية) .

فإذا خرجنا إلى النتيجة البديلة التي تعالج الحالات الأربع الثانية لوجدنا أن الحالتين ٥ ، ٦ تحتبران بشرط واحد فقط وهو (if largecar....) لذلك يخرج البرنامج من هذا الاختبار إلى الاختبار السابع مباشرة الذى يلى كلمة else .  
أى أن عدد الاختبارات التى قام بها البرنامج للوصول للحالة رقم ٨ هو ٣ اختبارات فقط .

بقى أن نكتب مضمون الرسائل الثمانية فى البرنامج حتى تكتمل معالته .  
والبرنامج شكل (٥ - ٢) يمثل هذه الصور النهائية وقد استخدمنا فيه الثوابت الحرفية للحفاظ على شكل البرنامج « المنطقى » !

```
program policy2(input,output);
```

البرنامج ٢

```
const p45 = 'policy loaded by 45 percent';
      p15 = 'policy loaded by 15 percent';
      p30 = 'policy loaded by 30 per cent';
      ok  = 'no loading';
      no  = 'no policy to be issued';
      p60 = 'policy loaded by 60 percent';
      p50 = 'policy loaded by 50 percent';
      p10 = 'policy loaded by 10 percent';
var over21, largecar, riskdriver : boolean;
    age, cc,convictions : integer;
```

```
begin
```

```
  read(age,cc,convictions);
  over21 := age >= 21;
  largecar := cc >= 2000;
  riskdriver := convictions >= 3;
```

وضع الرسائل  
في ثوابت حرفية

```
  if over21 then
```

```
    if largecar then
```

```
      if riskdriver then writeln(p45)
      else                writeln(p15)
```

```
    else
```

```
      if riskdriver then writeln(p30)
      else                writeln(ok)
```

```
  else
```

```
    if largecar then
```

```
      if riskdriver then writeln(no)
      else                writeln(p60)
```

```
    else
```

```
      if riskdriver then writeln(p50)
      else                writeln(p10)
```

```
end.
```

الحالات الأربع الأولى

الحالات الأربع الثانية



وهناك دائماً حل آخر .. فقد استخدمنا من قبل العبارة case لتمثيل البدائل المتعددة وهى بلا شك تصلح لتمثيل هذه الحالة مع استخدام تكنيك جديد فى إنشاء البرنامج وهو تداخل العبارات case بنفس الطريقة التى تتداخل بها العبارات الشرطية العادية . والبرنامج الآتى بعد شكل (٥ - ١٣) يوضح كيفية برمجة الحالات الأربع الأولى وقد تركنا الحالات الأربع الثانية كتمرين .

case over21 of

البرنامج ٣

true :

case largcar of

true :

case riskdriver of  
true : writeln(p45);  
false: writeln(p15)  
end;

false:

case riskdriver of  
true : writeln(p30);  
false: writeln(ok)  
end

end;

الحالات الأربع الأولى

false:

تستكمل بقية الحالات بنفس الأسلوب

end

شكل (٥ - ١٣)

(٥ - ٤) ملاحظات على بناء العبارات الشرطية المتداخلة :

رأينا أن العبارة الشرطية يمكن أن تحتضن بداخلها عدة عبارات شرطية أخرى بالصورة الآتية :

**if...then**

**if...then...**  
**else...**

**else**

**if....the...**  
**else...**

وترتيب العبارات بالصورة الموضحة لا يفيد الكمبيوتر في شيء وإنما يفيد المبرمج في ترتيب أفكاره وحتى يعرف « إلى أي if تنتمي كل else ؟ » !  
والبرنامج بهذه الصورة سهل القراءة . لكن الكمبيوتر لن يمانع أن يستقبل البرنامج كالآتي :

**if...then if...then...else...else if...then...else...**

وحتى لا يخلط الكمبيوتر في تبعية الكلمات else للكلمات if ، فإنه يعتبر أن أي else تتبع أقرب عبارة شرطية غير مكتملة . فهو كلما يصادف الكلمة else يبحث عن أقرب ثلاثي if - then - else (لاحظ الأسهم) .

ولنعتبر هذا المثال :

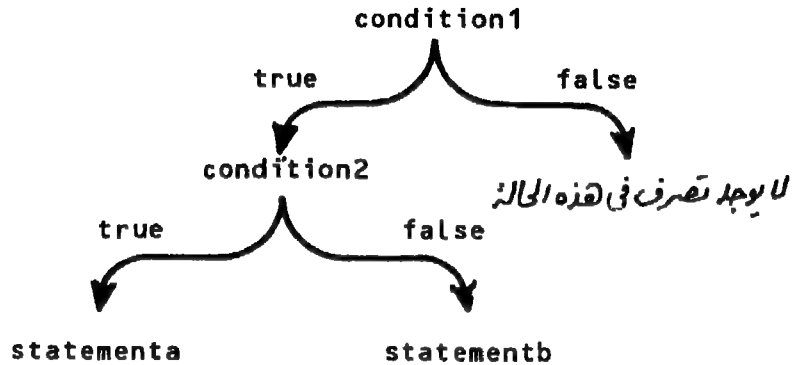
```

if condition1 then
    if condition2 then statementa
    else statementb
    
```

### شكل ( ٥ - ١٤ )

في هذا المثال فإن الكلمة else تتبع العبارة الشرطية الثانية (الكلمة if الثانية) وبذلك فإن البرنامج سوف يتبع المنطق الموضح في الشكل ( ٥ - ١٥ ) في تنفيذ

العبارات المختلفة وفقاً للشروط المختلفة .



### شكل (٥ - ١٥)

فإذا أردنا تمثيل شروطاً حقيقية فلنعتبر هذه الحالة :

```

if x > 50 then
  if y > 50 then writeln('both values > 50')
  else writeln('only the first value is > 50')
  
```

### شكل (٥ - ١٦)

في هذا المثال سوف يقوم الكمبيوتر بطبع رسائل معينة عندما تكون  $X > 50$  (مع جميع قيم  $Y$ ) أما الحالة التي تناظر  $X \leq 50$  فلن يقابلها أى تصرف .

وبصفة عامة فيمكنك استخدام عبارات البداية والنهاية (begin, end) لتنظيم البرنامج الذى يحتوى على عبارات شرطية متداخلة . ففى البرنامج السابق

يمكننا تغيير تبعية else للعبارة الشرطية الثانية ونقل تبعيتها إلى العبارة الشرطية الأولى باستخدام begin,end كآلاتي :

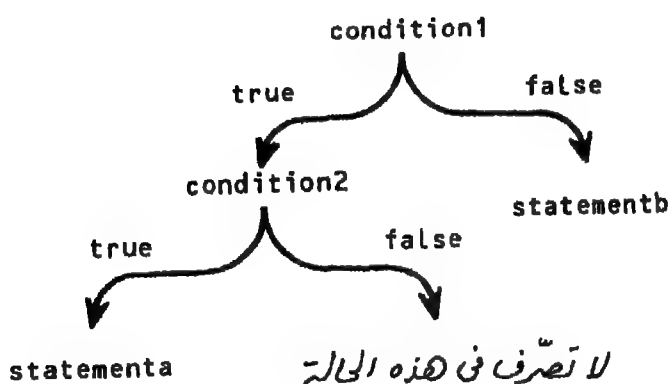
```

if condition1 then
begin
  if condition2 then statementa
end
else statementb

```

شكل (٥ - ١٦)

في هذه الحالة سوف يتغير المنطق الذي يتبعه البرنامج ويصبح كما في شكل (٥ - ١٧) .



شكل (٥ - ١٧)

فإذا عدنا للتمثيل بالأرقام نعتبر الصورة الجديدة للمثال :

```

if x > 50 then
begin
  if y > 50 then writeln('both values are > 50')
end
else writeln('the first value is not > 50')

```

### شكل (٥ - ١٨)

فإذا تتبعنا منطق الصورة الجديدة للبرنامج سوف نرى أن الحالة التي لا يقابلها أى تصرف من الكمبيوتر هي الحالة التي تكون فيها قيمة X أكبر من 50 وقيمة Y أصغر أو تساوى 50 .

### (٥ - ٥) مراجعة البيانات *data validation* :

عندما نبني برنامجاً يحتوى على العديد من البدائل باستخدام العبارة *case* فإنه من المحتمل أن يحتوى متغير الاختيار قيمة مختلفة عن قيم العناوين الموجودة بالبرنامج . وفى هذه الحالة فإن البرنامج يتوقف مرسلًا رسالة بالخطأ الذى حدث .

ولحماية البرنامج ضد مثل هذه الأحوال فإن يلزم إضافة عبارة شرطية تحبر البرنامج بالتصرف المطلوب منه إذا حدث مثل هذا الخطأ . وهذه الإضافة قد تكون بالصورة التالية :

<u>if</u>	<u>then</u>
<div style="border: 1px solid black; padding: 5px; display: inline-block;">case-statement</div>	
<u>else</u>	

### شكل (٥ - ١٩)

وبلى الكلمة if الشرط المطلوب توفره بحيث يضمن احتواء متغير الاختيار على القيمة الصحيحة دائماً ، وبلى الكلمة else التصرف المطلوب من البرنامج في حالة وجود قيمة مخالفة .

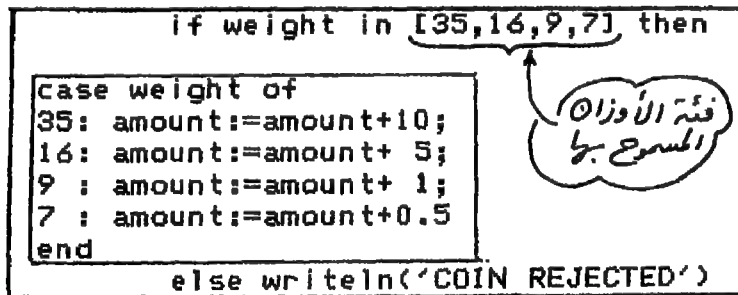
ولتوضيح استخدام هذا المنشأ نضرب مثلاً بماكينة العملة التي عالجناها من قبل في الباب الرابع .

مثال (٥ - ٦) فئات العملات :

تستقبل ماكينة العملة قطعاً من العملة ذات أوزان مختلفة ٣٥ ، ١٦ ، ٩ ، ٧ وهذه الأوزان تناظر العملات ١٠ ، ٥ ، ١ ،  $\frac{1}{4}$  قرشاً على الترتيب .. فإذا كانت العملة ذات وزن مختلف ، فعلى البرنامج أن يطرد العملة من الماكينة ويطبع الرسالة « COIN REJECTED » .

أما إذا كانت العملة هي أحد العملات المذكورة فيتم التعامل معها بالعبرة case كما سبق أن تعاملنا معها في الباب الرابع

وشكل (٥ - ٢٠) يوضح الفقرة المطلوب إضافتها لمراجعة وزن العملات .



شكل (٥ - ٢٠)

وكما نرى أن التعبير الشرطى قد تضمن الفئة **[35,16,9,7]** المميزة بالقوسين المربعين . واستخدام الفئة (set) يبنى عن عبارات بوليانية كثيرة مثل :

(weight = 35) or (weight = 16) or (weight) = 9) or  
(weight = 7 )

أما التعبير **weight in [35,16,9,7]** فيبنى باختصار واختصار ما إذا كانت قيمة المتغير weight هي أحد عناصر الفئة **[35,16,9,7]** .

واختبار الفئات يصلح لكثير من التطبيقات فعلى سبيل المثال يمكن اختبار قيمة المتغير y إذا ما كان يقع ما بين الواحد والعشرة (أى يتبى إلى فئة الأعداد من واحد إلى عشرة) كالتى :

**y in [1..10]**

والنقطتان المستخدمتان بداخل علامة الفئة تغنيان عن كتابة كل الأرقام بين الواحد والعشرة .

والتعبير السابق يكافئ تماماً التعبير المنطقى :

**(y >= 1) and (y <= 10)**

والقاعدة المحددة لئلا هذه الفئة أنها لا يجوز أن تحوى على أعداد حقيقية (سوف نعرض الموضوع تفصيلاً فى الباب الثامن) .

ولنعد الآن لمثال ماكينة العملة لنراه فى صورته الكاملة ، وفى هذا المثال فإن ماكينة العملة تستقبل قطع النقود وتجمعها حتى إذا بلغت ١٢٣ قرشاً أو زادت عن ذلك فإن البرنامج يطبع الرسالة : قبلت العملات COINS ACCEPTED ويحسب مقدار الباقي من النقود إن وجد . والبرنامج موضح بشكل (٥ — ٢١) .

## البرنامج

```

program coins(input,output);
var weight: integer;
    amount: real;
begin
    amount:=0;
    repeat
        read(weight);
        if weight in [35,16,9,7] then
            case weight of
            35: amount:=amount+10;
            16: amount:=amount+ 5;
            9 : amount:=amount+ 1;
            7 : amount:=amount+0.5
            end
        else writeln('COIN REJECTED')
        until amount >=123;
        write('COINS ACCEPTED');
        if amount > 123 then
            writeln('CHANGE DUE: ',amount-123)
        end.
    
```

مالية العملة

فئة الأوزان المسموح بها

رفض العملة

قبول العملات

حساب الباقي

## شكل (٥ - ٢١)

### (٥ - ٦) حماية البرنامج من البيانات الخاطئة :

في بعض الأحيان تتسبب البيانات غير الصحيحة في الحصول على نتائج غير صحيحة دون أن يتوقف البرنامج ودون أى رسالة تحذير من الكمبيوتر وهذه أخطر الحالات . لذلك يجب أن يتضمن البرنامج حماية ضد دخول مثل هذه البيانات . وطريقة الحماية تختلف بحسب نوعية البيانات الداخلة لكنه من



الشائع مع البيانات العددية على وجه الخصوص تحديد مدى معين للأرقام الداخلة للبرنامج فإذا خرجت عن هذا المدى أرسل البرنامج رسالة تحذير ! .

### مثال (٥ - ٧) كم تكلفك الأجازة ؟

يقوم هذا البرنامج بحساب تكاليف أجازتك وكل ما يطلبه منك من بيانات هو اليوم والشهر الذى تبدأ فيه الأجازة واليوم والشهر الذى تنتهى فيه الأجازة . ويقوم البرنامج بعد ذلك بحساب مدة الأجازة باستخدام هذه التواريخ وحساب تكلفة كل يوم باستخدام ٣ معدلات ثابتة فى البرنامج هى ٥٧ جنياً ، ٧٢ جنياً ، ٨٧ جنياً . والبرنامج يختار بين هذه المعدلات بحسب شهور السنة فشهور الصيف ٦ ، ٧ ، ٨ ، ٩ من الشهور المكلفة أما شهور الشتاء ١ ، ٢ ، ١١ ، ١٢ فهى شهور منخفضة التكاليف أما أشهر الربيع والخريف فهى معتدلة التكاليف . وبالطبع يمكنك تغيير هذه المعدلات بحسب المكان الذى تقضى فيه أجازتك .

```

program holiday(input,output) ;
const lowrate = 57;
      midrate = 72;
      peakrate = 87;

var day1, month1, day2, month2, duration,
    tillendofmonth, cost : integer;

```

تكاليف الإجازة

```

begin

```

```

  read(day1, month1, day2, month2) ;

```

```

  if (day1 in [1..31])
    and (day2 in [1..31])
    and (month1 in [1..12])
    and (month2 in [1..12])
    and (month1 <= month2)
  then

```

اختبار البيانات

```

begin
  if month1 = month2 then
    duration := day2 - day1 + 1
  else
    begin
      case month1 of
        2 : tillendofmonth := 28 - day1;
        9,4,6,11 : tillendofmonth := 30 - day1;
        1,3,5,7,
        8,10,12 : tillendofmonth := 31 - day1
      end;
      duration := day2 + tillendofmonth + 1
    end;

    case month2 of
      1,2,11,12 : cost := duration*lowrate;
      3,4,5,10 : cost := duration*midrate;
      6,7,8,9 : cost := duration*peakrate
    end;

    write (' cost of holiday is ', cost )
  end

```

معالجة البيانات

```

else write ('you have typed erroneous dates')

```

```

end.

```

رسالة خطأ عند إدخال بيانات غير صحيحة

### شكل (٥ - ٢٢)

أما الاختبار المطلوب إجراؤه على البيانات الداخلة للبرنامج فهو التأكد من أن الأرقام الممثلة للأيام تقع في الفئة [1..31] والأرقام الممثلة للشهور تقع في الفئة [1..12] وأن الشهر الذي تبدأ فيه الإجازة يساوي أو أكبر من

الشهر الذى تنتهى فيه الأجازة .

أما متغيرات البرنامج المستخدمة فهى كالآتى :

### (١) الثوابت :

lowrate	معدل تكلفة يوم أجازة فى الشتاء
midrate	معدل تكلفة يوم أجازة فى الربيع أو الخريف
peakrate	معدل تكلفة يوم أجازة فى الصيف

### (٢) المتغيرات :

day1	يوم نهاية الأجازة
day2	يوم نهاية الأجازة
month1	شهر بداية الأجازة
month2	شهر نهاية الأجازة
duration	مدة الأجازة
tillendofmonth	المدة حتى نهاية الشهر
cost	تكاليف الرحلة

### إضافة أخرى :

فى البرنامج السابق لا يتم معالجة البيانات قبل اختبارها وبالتالى فإن البيانات الصحيحة فقط هى التى تمر إلى الجزء من البرنامج المحصور داخل المربع . أما البيانات غير الصحيحة فينتج عنها توقف البرنامج وإرسال رسالة الخطأ المذكورة .

ولكن البرنامج الجيد هو الذى لا يتوقف لأى سبب من الأسباب بل دائماً يمنح فرصة أخرى لتصحيح الخطأ .

ويمكن إضافة الجزء التالي إلى البرنامج السابق حتى يكتسب هذه الخاصية ،  
حيث يتيه إلى الخطأ الذى وقع ويطلب منك إعادة إدخال البيانات .  
شكل (٥ - ٢٣) .

إضافة لبرنامج تكاليف الإجازة

```
read(day1,month1,day2,month2);
while not ((day1 in [1..31]) and
            (day2 in [1..31]) and
            (month1 in [1..12]) and
            (month2 in [1..12]) and
            (month1 <= month2)) do
begin
    write('you've made an error please retype');
    read(day1,month1,day2,month2);
end;
```

شكل (٥ - ٢٣)



## ■ مسائل على الباب الخامس :

س (٥ - ١) تقوم إحدى الشركات بتحليل يومى للمبيعات التى تمت خلال يوم كامل ويتضمن التحليل طبع قائمة بعدد السلع المباعة والتى يتعدى سعرها ١٠ جنيهات . ويتم هذا الحصر بالنسبة لجميع أقسام الشركة التى يبلغ عددها ٦٣ قسمًا . أكتب البرنامج الذى يقوم بهذا العمل علماً بأن المدخلات لهذا البرنامج عبارة عن قائمة بأسعار الأصناف المباعة فى اليوم المعين ، وتنتهى قائمة الأسعار برقم سالب مثل (1- ) .

س (٥ - ٢) فى مثال المراقبة الرقمية للأنتاج تم تصنيف أطوال المنتجات إلى نوعين :

- أطوال تقع فى النطاق المقبول ، وهى التى لا يزيد الفرق بينها وبين الطول القياسى (6.37) عن 0.1 .
- أطوال تقع خارج النطاق المقبول وهى الأطوال التى تتعدى الفرق السابق زيادة أو نقصاً .

والمطلوب تطوير البرنامج لكى يقوم بتصنيف الأطوال غير المقبولة إلى أطوال « أكبر من اللازم » وأطوال « أصغر من اللازم » .

س (٥ - ٣) أكتب برنامجاً يستقبل رقمى شهرين من نفس العام متبوعين برقم السنة ، ويطبع عدد الأيام من بداية الشهر الأول وحتى نهاية الشهر الثانى .

والمطلوب تطوير البرنامج السابق ليستقبل مجموعتين من الأرقام معبرة عن التاريخ (اليوم والشهر) بالصورة الآتية :

21 2

15 7

(بمعنى ٢١ فبراير و ١٥ يوليو)

ويلى هذه الأرقام الرقم الدال على السنة . والمطلوب حساب الفترة التى تقع  
بين هذين التاريخين بالأيام وطباعتها على الشاشة .



## الباب السادس

التعامل مع اللبنات  
(CHARACTER Manipulation)





## مفتح

حتى الآن ... قد تعاملنا مع الأعداد والبيانات العددية كمدخلات ومخرجات ومادة للمعالجة . وفي هذا الباب سوف نتلقى بالحرقية أو البيانات الحرقية التي تمثل أسماء الأشخاص والعناوين وأرقام التليفونات . وقد علمنا من قبل أن لغة باسكال تتعامل مع الحرفيات كلبينات مفردة (characters) .

فلغة باسكال القياسية ترى الحرفيات سواء كانت أسماء أو أرقام تليفونات أو نصوصاً أدبية ، كمجموعة من اللينات المتتالية . ومع ذلك فلنا عودة أخرى لموضوع الحرفيات بمعناها المعروف في الباب الثامن حيث نعرض الإمكانيات التي قدمتها الطرازات الجديدة للغة لخدمة معالجة الكلمات والنصوص .

## ٦ - ١) متغيرات اللبئات (character variables) :

علمنا من قبل أن اللبئات تنتمي إلى نمط خاص من الأنماط القياسية في اللغة والمسمى بنمط اللبئة (CHAR). ومتغير اللبئة لا يحتوى إلا على لبنة واحدة فقط ، قد تكون حرفاً أو رقماً أو علامة خاصة من العلامات التي على لوحة الأزرار .

وفي الأجزاء التالية سوف نعرف بعض الأمثلة لمعالجة اللبئات بالطرق المختلفة .

### مثال (٦ - ١) :

يقرأ هذا البرنامج ثلاث لبئات متتابعة ويطبعها في ترتيب معكوس . وقد أطلق الاسم "first" على المتغير الأول والاسم "second" على الثاني ، والاسم "third" على الثالث .

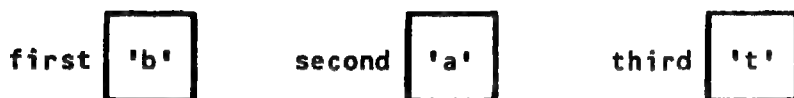
```
program reverse(input,output);
var first, second, third : char;
begin
  read(first, second, third);
  writeln(third, second, first)
end.
```

### شكل (٦ - ١)

عند تشغيل البرنامج سوف يتوقف عند العبارة read منتظراً إدخال اللبئات الثلاث . فإذا أدخلنا له ثلاثة أحرف مثل :

**bat**

فإنه يحفظها في الذاكرة في ثلاثة أوعية مستقلة يحمل كل منها اسم المتغير الخاص بها كما في الشكل (٦ - ٢) .



### شكل (٦ - ٢)

لذلك فعند تنفيذ أمر الطباعة سوف يسترجع البرنامج الحروف الثلاثة من الذاكرة بحسب ترتيب المتغيرات في عبارة الطباعة (وهو ترتيب معكوس) ويبدأ بطباعة المتغير الثالث ثم الثاني ثم الأول فتكون النتيجة هي الحروف :

**tab**

وبالطبع يجوز إدخال ثلاث علامات خاصة مثل :

\* + -

- + \*

فيصبح الخرج :

أو إدخال ثلاثة أعداد مثل :

**123**

**321**

فيصبح الخرج :

ولا يفوتنا ملاحظة أن هذه الأرقام لا يمكن إجراء عليها أية عمليات حسابية ، فهي ليست أكثر من لبنات متتابعة .

مثال (٦ - ٢) :

يقرأ هذا البرنامج ثلاثة لبنات ويطبعها بكل التباديل الممكنة . وللبرنامج خاصية هامة عند قراءة اللبنة فهو يتجاهل أية مسافات خالية قد تكون موجودة قبل أول لبنة من اللبنة الثلاث .

```

program anagrams(input,output);
var first, second, third : char;
begin
  repeat
    read(first)
  until first <> ' ';
  read(second, third);
  writeln(first,second,third, ' ', first,third,second);
  writeln(second,first,third, ' ', second,third,first);
  writeln(third,first,second, ' ', third,second,first)
end.

```

قراءة اللبنة الأولى  
مع تجاهل المسافات الخالية  
قراءة اللبنة الثانية والثالثة

شكل (٦ - ٣)

وعند تنفيذ هذا البرنامج سوف ندخل له ثلاث لبنات متتالية ولتكن : cat  
(يجوز ترك أية مسافات خالية قبل أول لبنة) . عندئذ يقوم البرنامج بطباعة التباديل الآتية :

```

cat  cta
act  atc
tca  tac

```

## (٦ - ٢) نهاية السطر *eol* (*end-of-line*) :

عندما ندخل بياناً من البيانات فإننا نضغط على الزر (ENTER) أو الزر (RETURN) في نهاية الإدخال لإعلام الكمبيوتر بأن البيان المطلوب جاهز على الإدخال والمعالجة . وما يحدث في حقيقة الأمر أن الضغط على زر إدخال البيانات يضيف إلى البيان المكتوب لبنة جديدة تسمى علامة نهاية السطر (*End-of-line marker*) . وحتى يمكن إدخال البيانات سطرًا بسطر فإن لغة باسكال تمدنا بالدالة *eol* (وهي اختصار العبارة *end-of-line*) ، وهذه الدالة من النوع المنطقي (أو البولياني) أى أنها تحتوى إما على القيمة *true* أو القيمة *false* . وهي تستخدم لاختبار وجود علامة نهاية السطر في البيان المُدخل عند قراءة كل لبنة تالية .

والبرنامج التالى يستقبل مجموعة من اللبنات ويقوم بحساب عدد اللبنات التى تمت كتابتها قبل الضغط على زر نهاية السطر (الزر ENTER أو RETURN) :

```
var nextchar : char;  count : integer;
.
.
count := 0;
while not eoln(input) do
begin
    read(nextchar);
    count := count + 1
end
```

## شكل (٦ - ٤)

\* ملاحظة : قد لا يعمل هذا البرنامج مع بعض الطرازات .

والدالة *eol* قد تكتب بالصورة *eol(input)* أو بالصورة المبسطة *eol* مع بعض الطرازات .

وعادة ما يتعامل برنامج باسكال مع علامة نهاية السطر كمسافة خالية وحتى نشاهد ذلك يمكن تجربة هذا البرنامج :

```
var ch1, ch2, ch3, ch4 : char;  
.  
.  
.  
read(ch1, ch2, ch3, ch4);  
write(ch1, ch2, ch3, ch4)
```

### شكل (٦ - ٥)

في هذا البرنامج سوف يتوقف التنفيذ عند العبارة read منتظر إدخال أربع لبنات متتالية ثم الضغط على زر نهاية السطر . فإذا كتبنا الحروف الآتية كمُدخل :

**abcd**

فإن البرنامج سيطبع نفس الحروف بموجب العبارة writeln :

**abcd**

لنجرّب الآن إدخال الحروف الأربعة بطريقة أخرى فنكتب الحرفين ab ثم نضغط زر نهاية السطر ثم نضغط على الحرف C ثم زر نهاية السطر ، في هذه الحالة نحصل على الجواب :

**ab c**

أى أن البرنامج قد اعتبر أن علامة نهاية السطر (الممثلة بالمسافة الخالية) هي أحد المدخلات الأربعة .

ولنجرّب أيضاً إدخال الحروف الأربعة abcd مع الضغط على الزر enter

(زر نهاية السطر) بعد كل حرف . عندئذ سنحصل على الجواب مباشرة بعد الضغطة الثانية قبل أن نكمل إدخال الحروف كلها وسيكون الجواب كآتي :

**a b**

فاللبنات الأربع المدخلة في هذه الحالة كانت :

- الحرف a .
- علامة نهاية السطر .
- الحرف b .
- علامة نهاية السطر .

ولو أننا كتبنا مجموعة كبيرة من الحروف بعد abcd سيظل الجواب دائماً هو abcd فعبارة القراءة في هذا البرنامج لا تتوقع إلا أربعة حروف فقط !  
وجرب هذا المثال :

<b>abcdefghijkl</b>	: المدخلات
<b>abcd</b>	: النتيجة

(٦ — ٣) نهاية الملف (eof) **end-of-file** :

تعتبر لغة باسكال أن كل المدخلات (input) والمخرجات (output) تأتي من ملفات (files) وتذهب إلى الملفات<sup>(١)</sup> . وقد تكون الملفات هي ملفات حقة بمعناها المتعارف عليه في لغات الكمبيوتر المختلفة بمعنى أنها مجموعة مترابطة من سجلات البيانات<sup>(١)</sup> (records) التي تحفظ على القرص المغنطيسي أو الشريط المغنطيسي ولكنها أيضاً قد تكون مجموعة من اللبنات المدخلة من لوحة الأزرار أو الخارجة إلى الشاشة أو جهاز الطباعة .  
لذلك فإن هناك نوعين من الملفات يتم فتحهما آلياً ببرنامج باسكال هما ملف

(١) للمزيد من التفاصيل عن الملفات يرجع لكتابنا « مدخلك إلى عالم الكمبيوتر » .

**الدخل input** ويستخدم لإدخال البيانات إلى البرنامج من جهاز الدخل القياسي (لوحة الأزرار في حالة الميكروكمبيوتر) ، وملف الخرج **output** وهو يستخدم لإخراج المعلومات إلى جهاز الخرج القياسي (الشاشة أو جهاز الطباعة) .

ولا حاجة بك لفتح هذه الملفات إذا أردت استخدامها ، أما الملفات الأخرى التي قد تكون في وسط آخر من أوساط الملفات مثل القرص المغنطيسي فتحتاج إلى إعلان مسبق .

ولسنا بصدد التعامل مع الملفات كموضوع مستقل الآن ، بل سنكتفى بالتعرض لملفات أجهزة الدخل والخرج القياسية وهي مجموعة اللبئات التي ترسل من لوحة الأزرار إلى البرنامج .

والدالة الجديدة التي تمدنا بها لغة باسكال هي دالة نهاية الملف (**eof**) والتي تستخدم لتعريف البرنامج بأن البيانات المدخلة قد بلغت نهايتها .

ودالة نهاية الملف قد تكتب بالصورة **eof** أو بالصورة الكاملة **eof(input)** .

والبرنامج التالي يقوم بحساب عدد اللبئات المدخلة بما في ذلك لبنة نهاية السطر (**eol**) باستخدام دالة نهاية الملف .

```
count := 0;
repeat
  read(nextchar);
until eof(input);
writeln('there are ', count, ' characters in the file.
```

*كرر العمل حتى تبلغ نهاية الملف*  
*عدد اللبئات*

شكل (٦ - ٦)

\* ملاحظة : دالة نهاية الملف قد يختلف استخدامها مع الطرازات المختلفة (يرجع لكتاب اللغة للجهاز) .



### مثال (٦ - ٣) كم مرة يتكرر الحرف في النص ؟

من التطبيقات المفيدة في معالجة اللبئات حساب عدد مرات تكرار حرف معين في سطر ما أو نص ما .

والبرنامج التالي يقرأ النص حرفاً حرفاً ويضعه في المتغير nextch ثم يقارنه بالحرف e المخزن في الثابت المسمى givench فإذا حدث التطابق أضاف واحداً إلى العداد noof occurrences وفي النهاية عندما يصل البرنامج إلى نهاية الملف يطبع محتوى العداد الذي يمثل عدد مرات التكرار .

```

program charcount(input,output);
const givench = 'e'; الحرف موضوع البحث
var noofoccurrences : integer; nextch : char;
begin
    noofoccurrences := 0; العداد
    repeat
        read(nextch);
        if nextch = givench then
            noofoccurrences := noofoccurrences + 1
    until eof(input); حتى نهاية الملف
    writeln('the character ', givench, ' appears ',
            noofoccurrences, ' times. ');
end. معدل تكرار حرف في نص

```

### شكل (٦ - ٧)

### (٦ - ٤) ترتيب اللبئات :

تأخذ اللبئات ترتيباً معيناً في تعاقبها ، وتمنح لغة باسكال رقماً لكل لبنة يمثل ترتيب هذه اللبنة في التابع فاللبنة الأولى تأخذ الرقم صفراً (0) والثابتة تأخذ الرقم 1 والثالثة 2 وهكذا ...

وتتابع الحروف الأبجدية من A إلى Z. تتابعاً متصلاً لا يتخلله فواصل أى أن الحرف A يأتي قبل B والحرف B يأتي قبل C وهكذا ... كما تتابع الأرقام أيضاً تتابعاً متصلاً. أما العلامات الخاصة فقد يختلف ترتيبها من طراز لآخر في لغة باسكال .

وكما نستخدم المؤثرات العلاقية مع الأعداد مثل :

$$x > y, z <= 2$$

يمكننا استخدام المؤثرات العلاقية أيضاً مع اللبئات فنقول :

$$'a' < 'b'$$

وإذا كان ch متغير لبنة فيجوز أن نكتب :

$$ch <= 'a'$$

$$ch >= 'z' \quad \text{أو}$$

وبذلك يمكن اختبار محتوى متغير اللبنة لمعرفة ما إذا كان يحتوي على حرف من الحروف باستخدام التعبير العلاقي :

$$(ch >= 'a') \text{ and } (ch <= 'z')$$

وباستخدام الفئات (sets) مع المؤثر in يمكن التعبير عن نفس المضمون بطريقة أفضل :

$$ch \text{ in } ['a'..'z']$$

كذلك لمعرفة ما إذا كان المتغير ch يحتوي على رقم يمكن استخدام التعبير العلاقي :

$$(ch >= '0') \text{ and } (ch <= '9')$$

أو باستخدام الفئات أيضاً :

**ch in ['0'..'9']**

مثال (٦ — ٤) تصنيف اللبئات :

يقوم هذا البرنامج باستقبال لبنة واحدة ثم يطبع رسالة على الشاشة ليخبرنا بنوع اللبنة هل هي حرف (alphabetic) أو رقم (numeric) أو ما عدا ذلك (not alphanumeric) .

والاختبار يتم على كل من فئة الحروف ['a'..'z'] وفئة الأرقام ['0'..'9'] فإذا كانت اللبنة لا تنتمي لإحدى الفئتين كانت لبنة خاصة (not alphanumeric) . شكل (٦ — ٨) .

```

program classifychar(input,output);
var character : char;
begin
  read(character);
  if character in ['a'..'z'] then
    writeln('that character is alphabetic.')
  else if character in ['0'..'9'] then
    writeln('that character is numeric.')
  else
    writeln('that character is not alphanumeric.')
end.

```

تصنيف اللبئات

فئة الحروف

فئة الأرقام

شكل (٦ — ٨)

مثال (٦ — ٥) كم حرفاً في الكلمة ؟

يستقبل هذا البرنامج كلمة ما ويحسب عدد حروفها .. وقد تضمن البرنامج شرطاً لتجاهل أى لبنة غير أبجدية (ليست حرفاً) قد تقع في بداية الكلمة ،

وكذلك فإن الكلمة المدخلة يعقبها لبنة غير أبجدية كعلامة لنهاية الإدخال .  
(ملاحظة : سوف نعرض في الباب السابع دالة جاهزة لحساب طول  
الكلمة مع موضوع الحرفيات (strings) .

```
program wordlength(input,output);
var nextchar : char; noofletters : integer;
begin
    repeat
        read(nextchar)
    until nextchar in ['a'..'z']:
        حتى هذه النقطة سوف يحتوى المتغير nextchar على الحرف الأول من الكلمة
        noofletters := 0;
        repeat
            noofletters := noofletters + 1;
            read(nextchar)
        until not (nextchar in ['a'..'z']);
        حتى هذه النقطة سوف يحتوى المتغير nextchar على اللبنة التالية للكلمة
        writeln('no. of letters: ', noofletters)
    end.
```

شكل (٦ - ٩)

مثال (٦ - ٦) النسبة المئوية لتكرار حرف :

أما البرنامج التالى هو يستقبل قطعة من نص ويقوم بحساب نسبة تكرار  
حرف معين في هذا النص . والحرف المطلوب حساب معدّل تكراره في هذا  
المثال هو الحرف "e" .

والخرج المتوقع من البرنامج يكون على الصورة :

**16% of letters were e-s**

والحرف s الذى يعقب الحرف e لزوم الجمع فى اللغة الانجليزية !

```

program letterfrequ(input,output);
const givenletter = 'e';
var noofletters, noofgivenletter : integer;
    character : char;
begin
    noofletters := 0; noofgivenletter := 0;
    repeat
        read(character);
        if character in ['a'..'z'] then
            begin
                noofletters := noofletters + 1;
                if character = givenletter then
                    noofgivenletter := noofgivenletter + 1
            end
        end
    until eof(input);
    writeln(noofgivenletter/noofletters*100,
            '% of the letters were ', givenletter, '-s.')
end.

```

الحرف المقصود  
عدد مرات تكرار الحرف  
عدد الحروف كلها  
حساب النسبة المئوية للتكرار

معدل تكرار حرف في نص

## شكل (٦ - ١٠)

ويترك للقارئ محاولة إنشاء برنامج لحساب عدد الكلمات فى النص كدرب .

(٦ - ٥) المزيد من الوسائل لمعالجة اللبئات :

(٦ - ٥ - ١) طباعة اللبئات على الشاشة :

يمكنك أن ترى الحروف الأبجدية مرتبة على الشاشة باستخدام هذه الشريحة الصغيرة :

```
var letter : char;  
.  
.  
.  
for letter := 'a' to 'z' do  
  write(letter)
```

البرنامج

abcdefghijklmnopqrstuvwxyz

التنفيذ

شكل (٦ - ١١)

وإذا استبدلنا عبارة التكرار « التصاعدي » بعبارة التكرار « التنازلي »  
downto فإن الحروف تطبع من الآخر إلى الأول أى من z إلى a .

```
.  
.  
.  
for letter := 'z' downto 'a' do  
  write(letter)  
.  
.  
.
```

البرنامج

zyxwvutsrqponmlkjihgfedcba

التنفيذ

شكل (٦ - ١٢)

## (٦-٥-٢) دوال اللبنة :

وفي الأجزاء التالية سوف نتعرض بتفصيل أكثر لخصائص اللبنة وترتيبها وطرق معالجتها بالدوال المختلفة .

• **الدالة pred** : تستخدم هذه الدالة لإيجاد اللبنة السابقة لللبنة أخرى في الترتيب ؛ وهي اختصار كلمة predecessor . فالعبرة :

**writeln(pred('b'))**

سوف تطبع الحرف a وهو الحرف السابق للحرف b المستخدم كدليل  
للدالة pred.

• **الدالة succ** : وهي الدالة المقابلة للدالة pred وهي مختصر كلمة successor ، وتستخدم لإيجاد اللبنة التالية لللبنة أخرى . فالعبرة :

**writeln (succ('b'))**

سوف تطبع الحرف c التالى للحرف b .

## مثال (٦ - ٧) رسالة مشفرة :

يقوم أحد رجال المخابرات بإرسال رسالة سرية إلى الكومبيوتر الرئيسى بمركز القيادة وحتى تتحقق عناصر السرية الكاملة فإنه يقوم بتشفير الرسالة باستخدام البرنامج الآتى الذى يستبدل كل حرف من الرسالة بالحرف التالى له فى الأبجدية كما يستبدل الحرف z بالحرف a . يقرأ البرنامج الرسالة الأصلية من ملف ثم يطبعها فى صورتها المشفرة .

```

program code(input,output);

var character : char;

begin
    repeat
        while not eoln(input) do
            begin
                read(character);
                if character in ['a'..'z'] then
                    if character = 'z' then write('a')
                    else write(succ(character))
                    else write(character);
                end;

                readln; writeln
            until eof(input)
        end.
    
```



### شكل (٦ - ١٣)

نلاحظ في هذا البرنامج ظهور عبارة قراءة جديدة هي readln وهي تستخدم لبدء سطر جديد من سطور الدخول ، تماماً كما تستخدم العبارة writeln لبدء سطر جديد من سطور الخرج .

\* الدالتان chr و ord :

ويسمى هذا العدد لترتيب اللبنة بالعدد الترتيبي (ordinal number) وتفيدنا الدالة (ord) في تحويل اللبنة إلى العدد الترتيبي المناظر لها . أما الدالة (chr) فهي تحول العدد الترتيبي إلى اللبنة المناظرة له .

أى أن :

**ord (char)**



تعطى هذه الدالة العدد الترتيبي المناظرة للينة (char) . ودليل هذه الدالة يكون دائماً لينة أو متغير لينة .

### chr (integer)

تعطى هذه الدالة اللينة المناظرة للعدد الصحيح الذى تؤثر عليه (integer) . والدليل لهذه الدالة يكون دائماً عدداً صحيحاً أو متغيراً صحيحاً . وترتيب اللينات قد يختلف من طراز إلى آخر من طرازات لغة باسكال .

مثال (٦ - ٨) :

البرنامج التالى يطبع اللينات من صفر إلى ٦٣ وأمام كل منها العدد الترتيبي الخاص بها (ordinal number) .

```
var ordinal : integer;
:
:
for ordinal := 0 to 63 do
  writeln('character ', chr(ordinal),
    ' has ordinal number ', ordinal)
```

الترتيب اللينة

شكل (٦ - ١٤)

أما شريحة البرنامج التالية فهي تطبع العدد الترتيبي لكل لينة من اللينات الأبجدية .

```
var ch : char;
:
:
for ch := 'a' to 'z' do
  writeln(ch, ' ', ord(ch))
```

العدد الترتيبي

شكل (٦ - ١٥)

أما شكل (٦ - ١٦) فيوضح البرنامج والتنفيذ على الكمبيوتر IBM ونرى أن التنفيذ يبدأ من الحرف m حتى الحرف z (بقدر ما تتسع الشاشة) ولعلنا نلاحظ أن ترتيب الحروف هو نفسه الكود آسكي (ASCII). حيث يبدأ الحرف a عند 97 وينتهي الحرف z عند 122

(بداية الحرف m)

```
var ch:char;
begin
  for ch:='m' to 'z' do
    writeln(ch,' ',ord(ch));
  end.
```

Running

```
m 109
n 110
o 111
p 112
q 113
r 114
s 115
t 116
u 117
v 118
w 119
x 120
y 121
z 122
```

تنفيذ البرنامج  
على الكمبيوتر IBM

>

شكل (٦ - ١٦)

ويمكن طباعة الحروف متجاورة باستخدام عبارة الطباعة write بدلاً من writeln كما في الشكل الآتي .

وفي هذا البرنامج استبدلنا الحروف الصغيرة بالحروف الكبيرة من A إلى Z  
(من 65 إلى 90).

**البرنامج**

```
var ch:char;
begin
  for ch:='A' to 'Z' do
    write(ch, ' ', ord(ch), ' ');
  end.
```

لائحة عبارة الطباعة

**التنفيذ**

Running

A	65	B	66	C	67	D	68	E	69	F	70	G	71	H	72	I	73	J	74	K	75	L	76	M	77	N
O	78	P	79	Q	80	R	81	S	82	T	83	U	84	V	85	W	86	X	87	Y	88	Z	89			

شكل (٦ - ١٧)

ولا أهمية كبيرة لمعرفة الأعداد الترتيبية بل يكفي معرفة الترتيب النسبي  
للحروف الأبجدية . والبرنامج التالي يطبع لك الحروف النوني (الحرف ذو  
الترتيب n) من الأبجدية عندما تدخل له قيمة المتغير n الذي يمثل ترتيب الحرف  
بالنسبة للحرف "a" إذا اعتبرنا أن الحرف a ترتيبه « الأول » .

**إيجاد الترتيب النسبي للحرف**

```
var n:integer;
begin
  readln(n);
  writeln('letter ', n, ' is ', chr(ord('a')+n-1));
end.
```

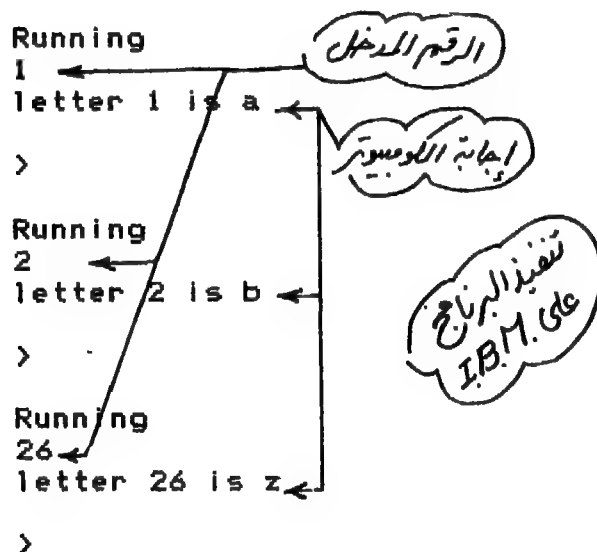
الترتيب

الحرف أو اللبنة

شكل (٦ - ١٨)

ونلاحظ في هذا المثال ظهور عبارة القراءة readln وفائدتها هنا فصل الخرج (إجابة الكومبيوتر) عن الدخول (البيان المدخل) حتى لا يظهران متجاورين على نفس السطر . ويمكن أداء نفس الغرض باستخدام read يليها writeln لتترك سطر خال .

والآتي بعد تنفيذ البرنامج على الكومبيوتر آى . بى . إم (IBM) حيث ندخل له رقماً من الأرقام فيخبرنا عن الحرف المناظر لهذا الرقم والممثل لترتيبه النسبى بالنسبة للحرف a .



شكل (٦ - ١٩)

## ■ تمرينات على الباب السادس :

س (٦ — ١) اكتب برنامجاً لتشفير رسالة سرّية بحيث يستبدل كل حرف من حروف الرسالة بالحرف الذى يليه بخمسة حروف فى الترتيب الأبجدي ، مع معاملة الحروف الأبجدية كنظام دائرى بمعنى أن الحرف a هو الحرف التالى للحرف z .

س (٦ — ٢) اكتب برنامجاً لقراءة ملف يحتوى على نص ويطبع عدد الحروف الأبجدية الموجودة بالنص كله كما يطبع رسالة أخرى لبيان عدد اللينات غير الأبجدية بما فى ذلك علامة نهاية السطر .

س (٦ — ٣) اكتب برنامجاً يحسب ويطبع عدد الكلمات فى جملة . ولنفرض أن الجملة مميّزة بنقطة فى نهايتها . احسب أيضاً عدد الكلمات المكونة من أربعة حروف واطبع عددها .

يمكنك الاستعانة بالمنطق الآتى :

● كرّر الآتى :

- ابحث عن وحدّد بداية الكلمة التالية .
- اقرأ الكلمة التالية واحسب عدد حروفها .
- زد عدد الكلمات بمقدار واحد .
- إذا كان عدد حروف الكلمة = ٤ فزد عدد الكلمات الرباعية بمقدار واحد .

● حتى تصل إلى النقطة '.' .

س (٦ — ٤) اكتب برنامجاً يقرأ جزءاً من أحد النصوص ويختبر مدى سلامة النص من جهة القاعدة الإملائية الآتية :

« فى أى كلمة يأتى الحرف (i) قبل الحرف (e) ما لم يسبقه الحرف (c) » .



## الباب السابع

### المصفوفات Arrays





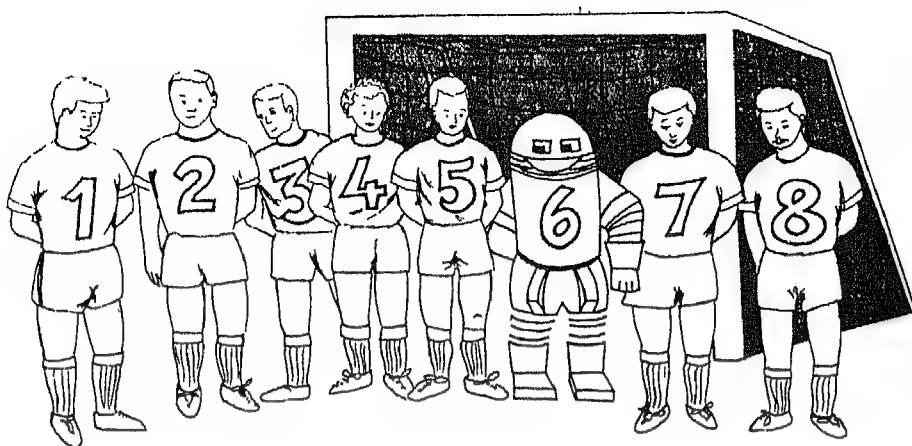
## مفتح

هل شاهدت مباراة لكرة القدم تلعب فيها فرق أجنبية ؟  
 وهل لاحظت أن المعلق الرياضى غالباً ما يستخدم الأرقام عندما  
 يتحدث عن اللاعبين الأجانب ؟ فيقول اللاعب رقم 2  
 واللاعب رقم 4 ؟ لماذا يستعين بالأرقام ؟  
 لأنها وسيلة أسهل لتمييز اللاعبين خاصة إذا كانت أسماءهم  
 جديدة علينا .

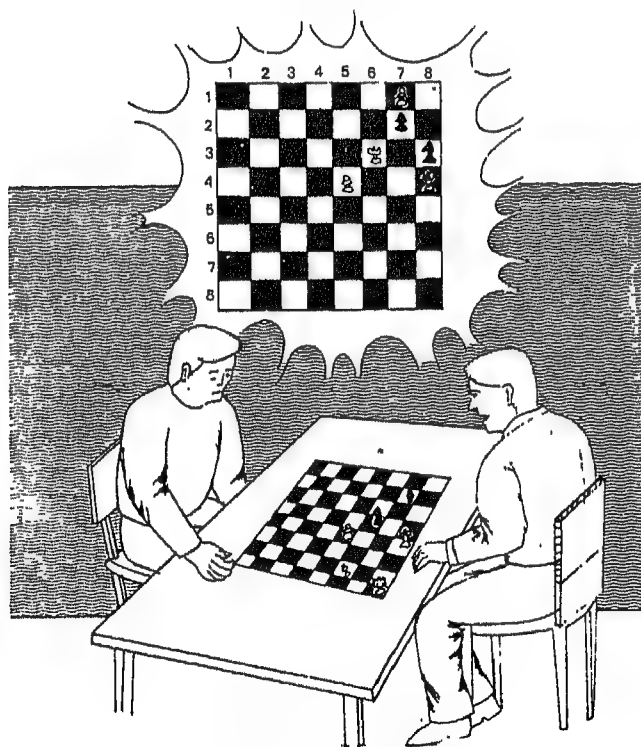
ومن وجهة نظر كومبيوترية : فإننا نقول أن لاعبي كرة  
 القدم يكوّنون مصفوفة واحدة . وكل لاعب منهم يعتبر عنصراً  
 من عناصر المصفوفة يميز برقمه بالنسبة لباقي عناصر مصفوفة  
 الفريق ، وهذا أسهل بكثير من تذكر أسماء اللاعبين .

كذلك فإذا أردت أن تصف مباراة للشطرنج فلا اعتقد أن  
 هناك بديلاً عن استخدام الأرقام لتمييز مربعات لوحة الشطرنج .  
 وفي هذه الحالة سوف يلزمك رقمان تمييز كل مربع : رقم يدل  
 على الصف (الأفقى) وآخر يدل على العمود (الرأسى) . فتقول  
 مثلاً أن الوزير الأبيض موجود بالمربع رقم (1.4) ، وهذا هو  
 الأسلوب المتبع فعلاً في لعب الشطرنج على الكومبيوتر حيث  
 يسألك البرنامج عن رقم المربع الذى تريد الانتقال إليه .  
 والشطرنج ينتمى إلى نوعية أخرى من المصفوفة تسمى المصفوفة  
 ذات البعدين .

والبديل الوحيد للتعبير عن مربعات الشطرنج بدون  
 استخدام أسلوب المصفوفات هو أن نبتكر إسماء لكل مربع  
 ويا له من عمل مزعج !



مصفوفة ذات بعد واحد



مصفوفة ذات بعدين

شكل (٧ - ١)

## **(٧ - ١) المتغيرات الدلييلة (subscripted variables) :**

تظهر الحاجة إلى استخدام المصفوفات عند التعامل مع البيانات  
المجدولة مثل درجات تلميذ في مختلف المواد الدراسية أو درجات  
فصل دراسي بأكمله .

وتعرّف المصفوفة عموماً بأنها :

« مجموعة مفهرسة من البيانات »

فإذا أردنا التعبير عن درجات أحد الطلبة في المواد الدراسية  
المختلفة فإنه من الممكن أن نعطي كل مادة اسم متغير مستقل مثل :

**ARABIC , ENGLISH, MAT,....**

ولكن من الأفضل أن نختار اسماً عاماً للمواد الدراسية وليكن :

### **SUBJECT [I]**

حيث يسمى المتغير I بالدليل وهو يأخذ قيمة مختلفة ، ويمكن  
في هذه الحالة أن تمثل درجات المواد المختلفة كالآتي :

### **SUBJECT [1], SUBJECT[2] , SUBJECT [3],...**

وتسمى هذه النوعية من المتغيرات بالمتغيرات الدلييلة لأنها  
تعتمد على دليل عام هو I فإذا أخذ الدليل القيمة 1 فذلك يدل  
على درجة اللغة العربية ARABIC وإذا أخذ القيمة 2 دل المتغير على  
درجة الانجليزية ENGLISH وهكذا وهذا يغني عن اختيار أسماء  
متغيرات كثيرة .

وتشكل جميع عناصر المتغير الدليلي SUBJECT[1] مصفوفة ذات بعد واحد

. (one dimensional array)

وقد مررنا بهذه النوعية من البيانات في الأجزاء السابقة وقد أطلقنا عليها اسم الفئة (set) مثل فئة الحروف الأبجدية ['A'..'Z'] وفئة الأرقام [0..9] والحقيقة أنه لا فرق بين الاسمين فالحروف الأبجدية هي مصفوفة ذات بعد واحد وهي عناصر فئة الحروف الأبجدية . كذلك فإن لاعبي كرة القدم ينتمون جميعاً لفئة واحدة هي الفريق .

فإذا أتينا إلى حالة تمثيل نتائج مجموعة كبيرة من الطلبة فإننا عادة ما نكتب النتيجة في صورة جدول كالوضح في شكل (٧ - ٢) الذي نطلق عليه المصفوفة ذات البعدين<sup>(١)</sup> (two dimensional array) .

*MARK [ 2 , 4 ]*

اسم التلميذ	اللغة العربية	الدين	اللغة الإنجليزية	اللغة الفرنسية	الرياضة	الطبيعة
محمد سمير	80	90	82	75	98	79
رائي محمد	80	95	90	76	100	81
سالى أسامة	75	80	90	83	99	85
مها حسن	79	85	85	82	85	80
عمرو الحسيني	85	74	92	80	100	65

شكل (٧ - ٢)

(١) كما يطلق عليها الرياضيون الاسم (Matrix) بمعنى مصفوفة أيضاً . وإذا كان عدد الأعمدة مساوياً لعدد الصفوف فإنها تسمى المصفوفة المربعة (square matrix) .

وقد مررنا بمثل هذه الحالة في الأبواب السابقة ولكننا عاجزاً بطرق  
بدائية كتدريب على استخدام الحلقات التكرارية . فإذا تعاملنا معها  
كمصفوفة فإن لغة باسكال تمدنا بتسهيلات جديدة لبرمجة المسألة .  
ويمكننا عندئذ أن نمثل درجة أى تلميذ عام في أى مادة عامة بالمتغير :

### MARK[I,J]

وقد استخدمنا هنا دليلين I , J حيث يدل I على اسم التلميذ (الصف) وتدل J  
على المادة (العمود) ، وبذلك تكون MARK[1,1] هي درجة التلميذ الأول في  
المادة الأولى أما mark[1,5] فهي درجة التلميذ الأول في المادة الخامسة ،  
كذلك فإن MARK[4,2] هي درجة التلميذ الرابع في المادة الثانية ..  
وهكذا ..

وكما نرى فإنه في حالة تمثيل المصفوفة ذات البعدين فإن المتغير الدليل يحتاج  
إلى دليلين حتى يعبر عن أى عنصر من عناصر المصفوفة .

والدليل في لغة باسكال لا يشترط أن يكون حرفاً واحداً مثل I, J ولكن  
ينطبق عليه ما ينطبق على أسماء البيانات من قواعد . فالأدلة الآتية جائزة أيضاً :

### mark[student, subject]

والمصفوفة ذات البعدين هي أيضاً فئة لأنها تحتوى على عناصر تنتمى انتماءً  
واحداً . وكلمة الفئة هي كلمة مألوقة في الرياضيات الحديثة .

### (٧ - ٢) الإعلان عن المصفوفات في البرنامج :

يتم الإعلان عن المصفوفات بطريقة خاصة تؤخذ فيها عدة اعتبارات . فقد  
تحتوى المصفوفة على أرقام حقيقية أو صحيحة وقد تحتوى على لبنات وهذا  
يستلزم الإعلان عنه في بداية البرنامج .

كما أن الإعلان عن المصفوفة يعنى حجز عدد من الأماكن في الذاكرة يتسع لعناصر المصفوفة أى يساوى بُعد المصفوفة . لذلك يلزم التنويه عن نوعية المتغيرات الدلالية وعن عددها .

ولنضرب مثلاً بالمصفوفة الآتية التي تعبر عن أسعار ثمانية أصناف مختلفة من السلع شكل (٧ - ٣) ، وسوف نفترض أن اسم المتغير الدليل العام هو :

**priceof[i]**

المحتوى ( السعر )	اسم المتغير
5.77	priceof[1]
3.15	priceof[2]
2.50	priceof[3]
16.33	priceof[4]
2.50	priceof[5]
13.45	priceof[6]
.86	priceof[7]
1.98	priceof[8]

شكل (٧ - ٣)

للإعلان عن مثل هذه المصفوفة في بداية البرنامج نكتب السطر التالي :

**var priceof : array [1..8] of real;**

لاحظ الكلمات الجديدة التي تحتها خط في هذا الإعلان !

إن هذا الإعلان يحتوى على كلمة array بمعنى مصفوفة ويحتوى على أبعاد المصفوفة أو عدد عناصرها وهو ثمانية عناصر ، كما يحتوى على نوع المتغيرات وهو النوع الحقيقي (real) .

والطريقة التي تستخدم للتعبير عن بُعد المصفوفة طريقة مميزة فهي تحتوى على رقم أول عنصر (1) وآخر عنصر (8) والنقطتان بينهما تعنيان بقية عناصر الفئة . وتتميز هذه الطريقة بإمكان إنشاء مصفوفة تبدأ من أى عنصر آخر خلال العنصر رقم 1 ، فيجوز أن نبدأ بالعنصر الخامس مثلاً فيصبح بعد المصفوفة [5..8] وهى فى هذه الحالة تحتوى على ٤ عناصر فقط .

أما الإعلان عن المصفوفة ذات البعدين فهو يتم بطريقة مماثلة فالمصفوفة الموضحة شكل (٧ - ٤) تحتوى على ٨ صفوف و ١٠ أعمدة أى أن أبعادها ١٠ × ٨ بمعنى أنها تتسع لثمانين عنصر . فإذا أطلقنا على المتغير العام لهذه المصفوفة الاسم .

**popmap[row,column]**

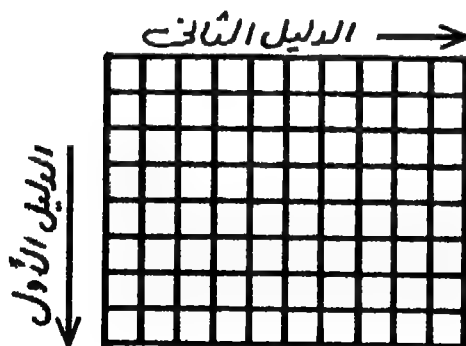
حيث يمثل الدليل row رقم الصف .

ويمثل الدليل column رقم العمود .

فإن الإعلان يكون بالطريقة الآتية :

**var popmap : array [1..8,1..10] of integer;**

وهى مصفوفة من الأعداد الصحيحة .



شكل (٧ - ٤)

## (٧ - ٣) معالجة المصفوفات ذات البعد الواحد

### One-dimensional arrays

بعد الإعلان عن المصفوفة يصبح لدينا عدد من المتغيرات الدلالية مساو لعدد عناصر المصفوفة ولنا أن نجرى على هذه المتغيرات ما نشاء من عمليات المعالجة التي نجرىها على المتغيرات العادية من التخصيص والقراءة والطباعة إلى آخره ، ولكن علينا ألا ننسى أثناء كتابة البرنامج أن نكتب الدليل المناسب للمتغير كالمثال الآتي :

مثال (٧ - ١) :

```
read(price[1], price[2], price[3], price[4]);
read(price[5], price[6], price[7], price[8]); ← قراءة
writeln('the price of item 2 is ', price[2]); ← طباعة
writeln('the price of item 4 is ', price[4]);

writeln('the price difference between items 1 and 2 is ',
      abs(price[1] - price[2]));

difference := price[1] - price[2]; ← معالجة حسابية
if difference > 0 then
  writeln('item 1 costs ', difference,
    ' more than item 2.')
else if difference < 0 then
  writeln('item 2 costs ', -difference,
    ' more than item 1.')
```

### شكل (٧ - ٥)

وللحلقات التكرارية فائدة كبيرة في معالجة المصفوفات لا سيما في القراءة والطباعة . فبدلاً من قراءة كل عنصر باستخدام اسمه الخاص يمكن استخدام الاسم العام بداخل حلقة تكرارية بحيث يكون الدليل هو نفسه عدّد الحلقة التكرارية كالمثال الآتي : شكل (٧ - ٦) :



مثال (٧ - ٢) :

```

program reverse(input,output);

var number : array [1..10] of integer;
    position : integer;

begin
    العداد
    for position := 1 to 10 do
        read(number[position]);
    الدليل
    for position := 10_downto 1 do
        writeln(number[position])
    end.

```

شكل (٧ - ٦)

والبرنامج السابق يقوم بقراءة مصفوفة ذات بعد واحد مكونة من عشرة عناصر من الأعداد الصحيحة ، ثم يطبع المصفوفة بطريقة عكسية أى بدءاً من العنصر العاشر وحتى العنصر الأول .

مثال (٧ - ٣) :

في بعض الأحيان تصبح عملية التخصيص ذات فائدة لا سيما عندما تكون محتويات المتغيرات ثابتة . والبرنامج التالي شكل (٧ - ٧) يوضح كيفية حساب سعر كمية معينة من الأصناف المباعة . فهو يحتوى على أسعار أربع سلع مختلفة تم تخصيصها للمتغيرات الدليلية الممثلة لأسعار الأصناف ... **priceof[1], priceof[2]** ويستقبل البرنامج الكمية المباعة quantity ثم يحسب سعرها ويطبعه .

```

program prices(input,output);

var priceof : array [1..4] of real;
    itemno, quantity : integer;
    total : real;

begin

    priceof[1] := 5.77;
    priceof[2] := 3.15;
    priceof[3] := 2.50;
    priceof[4] := 1.35; } التخصيص  
لأسعار الأصناف

    total := 0;

    for itemno := 1 to 4 do
    begin
        read(quantity); حساب السعر الكلي
        total := total + quantity*priceof[itemno]
    end;

    writeln('the total cost is ',total)
end.

```

### شكل (٧ - ٧)

وبالطبع فإن طريقة التخصيص ليست هي الطريقة المثلى حيث أن أى تغيير يطرأ على سعر السلعة سوف يستلزم إصلاح البرنامج نفسه . أما المتبع فى مثل هذه الحالات أن تحفظ أسعار السلع فى ملف (file) وتتم القراءة منه آلياً . وفى حالة تغيير الأسعار فإن برنامجاً آخر يتولى تحديث الملف .

مثال (٧ - ٤) :

رأينا من قبل المصفوفة ذات الأعداد الحقيقية أو الأعداد الصحيحة ، وهذا المثال يعرض نموذجاً لمصفوفة اللبئات . شكل (٧ - ٨) .

ونلاحظ فى هذا البرنامج أن عدد عناصر المصفوفة المعلن عنها هو ٢٠

عنصراً . لكن البرنامج يقرأ ويطبّع عدداً من العناصر يتحدد بالمتغير  $n$  أثناء تنفيذ البرنامج .

فالبرنامج يبدأ بقراءة قيمة المتغير  $n$  بموجب العبارة **readln(n)** فإذا أدخلنا له العدد 5 مثلاً فسوف يقوم البرنامج بمعالجة خمسة عناصر فقط من المصفوفة . ويجوز إدخال أى رقم كقيمة للمتغير  $n$  بشرط ألا يزيد عن 20 وإلا يفشل البرنامج .

```
program word(input,output);
var letter : array [1..20] of char;
    n, i : integer;
begin
    readln(n);
    for i := 1 to n do
        read(letter[i]);

    writeln;
    for i := n downto 1 do
        write(letter[i])

end.
```

### شكل (٧ - ٨)

مثال (٧ - ٥) تجميع وفرز الأصوات :

نعود مرة أخرى إلى معالجة الانتخابات ونلتقى في هذا المثال بستة من المرشحين ويتولى البرنامج فرز الأصوات التي حصل عليها المرشحون وإعلان رقم المرشح الفائز . شكل (٧ - ٩) .

```
program votes(input,output);
```

تجميع وفرز الأصوات

```
var candidate, mostvotesofar, winner : integer;
    votesfor : array [1..6] of integer;
```

```
begin
```

شحن المتغيرات بقيمة ابتدائية صفرًا

```
for candidate := 1 to 6 do
    votesfor[candidate] := 0;
```

```
read(candidate);
repeat
    votesfor[candidate] := votesfor[candidate] + 1;
    read(candidate);
until candidate = -1;
```

إدخال البيانات

نهاية البيانات

```
mostvotesofar := 0;
for candidate := 1 to 6 do
    if votesfor[candidate] > mostvotesofar then
        begin
            mostvotesofar := votesfor[candidate];
            winner := candidate;
        end;
```

فرز الأصوات

```
writeln('winner is candidate no. ',winner);
end.
```

رقم المرشح الفائز

شكل (٧ - ٩)

وقد استخدمت في هذا البرنامج الأرقام من 1 إلى 6 كأرقام مرجعية للدلالة على كل مرشح. وقد استخدم المتغير العام `votesfor[candidate]` للتعبير عن المتغيرات الدليلية لمصفوفة المرشحين فمثلاً:

`votesfor[1]` هو متغير تجميع الأصوات للمرشح الأول

`votesfor[2]` هو متغير تجميع الأصوات للمرشح الثاني

وهكذا ...

يبدأ البرنامج بالإعلان عن المتغيرات المستخدمة وهي :

( ١ ) متغيرات صحيحة (integer) :

- candidate رقم المرشح (وهو يستخدم كدليل لمتغيرات المصفوفة) .
- mostvotesofar وعاء لتجميع أعلى نسبة أصوات لأي مرشح .
- winner رقم المرشح الفائز .

( ٢ ) المصفوفة (array) :

تم الإعلان عن مصفوفة واحدة ذات عناصر ستة وهي : votesfor وتستخدم المتغير candidate للدلالة على رقم المرشح .

وينقسم البرنامج إلى ثلاثة أجزاء أساسية تم وضعها بداخل مستطيلات للإيضاح :

● الجزء الأول (المستطيل الأول) :

لشحن جميع المتغيرات الدليلية بالقيمة الابتدائية « صفر » وهذه عادة حميدة يجب إتباعها مع كل البرامج .

● الجزء الثاني (المستطيل الثاني) :

لإدخال الأصوات التي حصل عليها المرشحون ، والنظام المتبع هنا هو إدخال رقم المرشح الذي حصل على صوت ، فالرقم 1 يدل على أن المرشح الأول حصل على صوت واحد ، والرقم 5 يدل على أن المرشح الخامس حصل على صوت واحد وهكذا ..

وبمجرد إدخال صوت واحد فإنه يضاف إلى الأصوات السابقة التي حصل عليها نفس المرشح .

فإذا انتهت جميع الأصوات ندخل إلى البرنامج العدد (1 -) للدلالة على انتهاء

إدخال البيانات .

أما الجزء الثالث (المستطيل) : فهو يستخدم لمقارنة محتويات العناصر الستة للمصفوفة لإيجاد أكبر نسبة من الأصوات حصل عليها أى مرشح .

والمنطق المستخدم للمقارنة منطق معروف ، وهو مقارنة كل متغير مع الوعاء mostvotessofar فإذا كان محتوى المتغير أكبر من محتوى الوعاء يتم وضع محتوى المتغير في الوعاء ، وبذلك فإن الوعاء يحتوى دائماً على أكبر رقم . أما دليل المتغير الذى تجرى عليه المقارنة فهو يوضع في الوعاء winner . وفي نهاية عملية المقارنة سوف يحتوى الوعاء winner على رقم المرشح الذى حصل على أعلى نسبة أصوات .

مثال (٧ — ٦) عدد أيام الإجازة :

عاجلنا من قبل موضوع « كم تكلفك الإجازة ؟ » في الباب السادس . وفي هذا المثال نضيف إلى هذا المنطق معالجة جديدة باستخدام المصفوفات لحساب الفترة بالأيام بين تاريخين متتاليين يتكون كل منهما من اليوم والشهر .

```

program days(input,output);

var month, daystogo, firstday,
    firstmonth, secondday, secondmonth : integer;
    daysin : array [1..12] of integer;

begin
    for month := 1 to 12 do
        case month of
            2: daysin[month] := 28;
            9,4,6,11: daysin[month] := 30;
            1,3,5,7,8,10,12: daysin[month] := 31
        end;
        read(firstday,firstmonth,secondday,secondmonth);
        if firstmonth = secondmonth then
            daystogo := secondday - firstday
        else
            begin
                daystogo := daysin[firstmonth] - firstday;
                for month := firstmonth + 1 to secondmonth - 1 do
                    daystogo := daystogo + daysin[month];
                daystogo := daystogo + secondday
            end;
        writeln('days to go:', daystogo)
    end.

```

عدد الأيام في كل شهر

التاريخان في نفس الشهر

التاريخان في شهرين مختلفين

عدد الأيام

شكل (٧ - ١٠)

## متغيرات البرنامج :

### (١) المتغيرات الصحيحة :

● month : أحد شهور السنة وهو يأخذ القيم من 1 إلى 12  
ويستخدم كدليل للمصفوفة daysin .

● daystogo : عدد الأيام في الفترة المطلوبة بين التاريخين (أيام  
الإجازة)

{	رقم اليوم الأول	: firstday	●
	رقم الشهر الأول	: firstmonth	●
{	رقم اليوم الثاني	: secondday	●
	رقم الشهر الثاني	: secondmonth	●

### (٢) المصفوفات :

● daysin مصفوفة صحيحة تحتوي على ١٢ عنصراً

والتغير الدليلي العام لهذه المصفوفة هو **daysin[month]** ويعني عدد  
الأيام في شهر ما مثل :

عدد أيام شهر فبراير **daysin[2] = 28**

عدد أيام شهر مارس **daysin[3] = 31**

والبرنامج يقوم بحساب عدد الأيام بين التاريخين وفقاً لحالتين :

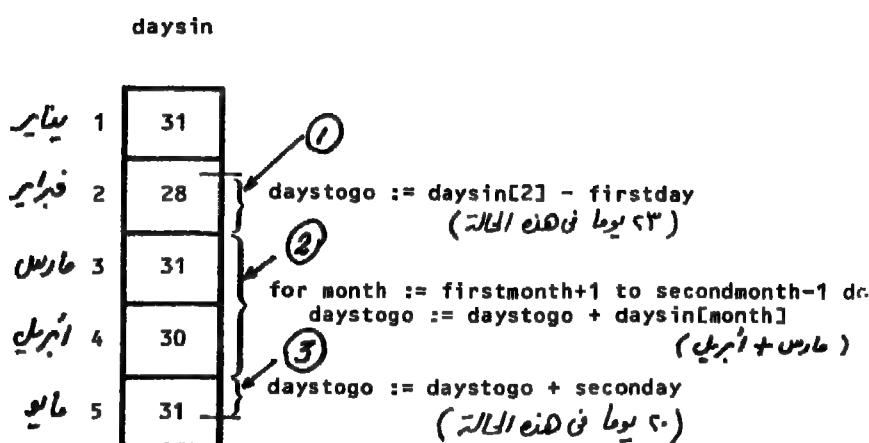
(١) إذا كان التاريخان في نفس الشهر فيتم ذلك بطرح رقم اليوم الأول من  
رقم اليوم الثاني .

(٢) إذا كان الشهران مختلفين فإن يتم جمع ثلاثة أجزاء معاً لتكوين عدد الأيام  
المطلوبة . فإذا كان التاريخان على سبيل المثال هما :



5 2 (٥ فبراير)  
20 5 (٢٠ مايو)

فإن المدة بينهما تحسب بإضافة الأجزاء الثلاثة الموضحة في شكل (٧ - ١١) .



شكل (٧ - ١١)

#### (٧ - ٤) الرسم البياني histograms :

عاجلنا من قبل موضوع رسم الأشكال المختلفة باستخدام الحلقات التكرارية والرسم البياني أو المستوجرام histogram يعنى تحويل المعلومات الناتجة من الكمبيوتر من صورتها العددية إلى رسم بياني يسهل فهمه بإلقاء نظرة عامة عليه بدلاً من فحص الأرقام . والرسم البياني ذو أهمية كبيرة في الموضوعات الإحصائية .

total sales for each dept.(div. by 10)

المبيعات الكلية للأقسام المختلفة	
dept. no.	1 *****
رقم القسم	2 *****
	3 *****
	4 *****
	5 *****
	6 *****
	7 *****
	8 *****

شكل (٧ - ١٢)

فإذا ألقينا نظرة عامة إلى شكل (٧ - ١٢) الذى يوضح المبيعات الكلية  
لثمانية أقسام فى شركة ما ، فسوف ندرك على الفور أن القسم رقم ٧ هو الذى  
حقق أعلى نسبة مبيعات وأن القسم رقم ٨ هو أقل الأقسام فى المبيعات .

ولعل المشاهد لهذا الرسم يستطيع الخروج بانطباع سريع عن نشاط كل  
قسم من الأقسام بالنسبة لبقية الأقسام الأخرى ، وهذا لا يتحقق عادة إذا كان  
الخروج عبارة عن أرقام تحتاج للفحص والمقارنة .

والبرنامج التالى يوضح كيفية الحصول على هذا الرسم البياني باستخدام  
برنامج باسكال .

مثال (٧ - ٧) تحليل المبيعات بالرسم البياني :

```

program salesanalysis2(input,output);
var dept, col, deptsales, salesin10s : integer;
begin
  writeln('total sales for each dept.(div. by 10)' : 48);
  writeln('-----' : 48);
  for dept := 1 to 8 do
  begin
    if dept = 4 then write('dept. no. 4 ');
    else
      , write(dept:11, ' ');
    read(deptsales);
    salesin10s := deptsales div 10;
    for col := 1 to salesin10s do
      write('*');
    writeln
  end
end.

```

البرنامج

طباعة أرقام الأقسام

مقياس الرسم

الرسم

تحليل المبيعات بالرسم

## شكل (٧ - ١٣)

والمغيرات المستخدمة في هذا البرنامج هي :

- dept : رقم القسم (العداد).
  - col : رقم العمود الرأسى في خانات الطباعة (العداد).
  - deptsales : مبيعات القسم
  - salesin10s : مبيعات القسم مقسومة على مقياس الرسم (10).
- وكلها متغيرات صحيحة .

ومقياس الرسم ضرورى في مثل هذه البرامج حتى لا يخرج الرسم عند حدود الورقة ويترك تقديره للمبرمج بحسب اتساع خط الطباعة .

هل لاحظنا أن البرنامج لم يستخدم المصفوفات ؟

في الواقع أنه لم تكن هناك حاجة لاستخدامها فاليان تتم قراءته ومعالجته فور إدخاله .

ولكن في بعض الأحيان قد يصبح استخدام المصفوفات ضرورياً ولنفرض هذه الحالة عندما يكون اتساع خط الطبع ٦٠ لبنة وعندما لا يمكن التنبؤ إذا ما كانت أعلى نسبة للمبيعات سوف تتعدى هذا الاتساع أم لا . في هذه الحالة يلزم إجراء معالجة بهدف معرفة مقياس الرسم المناسب . فيصبح البرنامج كالاتي :

```

program salesanalysis3(input,output);
const lengthofline = 60; مصفوفة الأقسام
var salesfor : array [1..8] of integer;
    dept, col, maxsofar, scaling, scaledtotal : integer;

begin
    maxsofar := 0; تحديد أعلى نسبة مبيعات
    for dept := 1 to 8 do
    begin
        read(salesfor[dept]);
        if salesfor[dept] > maxsofar then
            maxsofar := salesfor[dept]
        end;
        تحديد مقياس الرسم
        scaling := 1 + maxsofar div lengthofline;

        for dept := 1 to 8 do
        begin
            الرسم
            write(dept, ' ');
            scaledtotal := salesfor[dept] div scaling;
            for col := 1 to scaledtotal do
                write('*');
            writeln
        end
    end.

```

*تحليل المبيعات بالرسم*

شكل (٧ - ١٤)

## (٧ - ٥) شحن المتغيرات بقيمة ابتدائية initialization :

لعلنا لاحظنا في البرامج السابقة جميعاً أنه يتم شحن المتغيرات دائماً بقيمة ابتدائية صفراً (ما لم تكن هناك قيمة ابتدائية خلاف الصفر) ، وهذه الخطوة تعتبر ضرورية في البرنامج لأن خانات الذاكرة في بعض الأحيان قد تحتوى على « مخلفات » من البيانات ولنر هذا المثال الذى نقرأ فيه مصفوفة عددية ونشحنها بالقيمة صفراً ثم نطبعها شكل (٧ - ١٥) .

```
program chararrays(input,output);
VAR ch: array[1..20] of integer;
    i: integer;
```

البرنامج

BEGIN

```
for i:=1 to 10 do ← الشحن الابتدائي
ch[i]:=0;
```

```
writeln;
for i:=1 to 10 do ← الطباعة
writeln(ch[i]);
```

END.

Running

```
0
0
0
0
0
0
0
0
0
0
0
```

عناصر المصفوفة

التنفيذ

شكل (٧ - ١٥)

لنحذف الآن عبارة شحن المتغيرات بقيمة ابتدائية يوضعها بين القوسين  
 ""[]"" اللذين يحولانها إلى ملاحظة ، ولنر النتيجة .

```
program cchararrays(input,output);
VAR ch: array[1..20] of integer;
    i: integer;
```

BEGIN

```
{ for i:=1 to 10 do
  ch[i]:=0.0; }
```

← عبارة  
محذوفة  
(لاحظ أقواس التعليقات)

```
writeln;
for i:=1 to 10 do
  writeln(ch[i]);
```

END.

Running

```
0
0
0
0
0
0
0
0
0
-32768
358
```

مخلفات  
بيانات!

شكل (٧ - ١٦)

ما هي النتيجة تحتوي على مخلفات لا معنى لها من الأرقام .  
 ولو قمنا بطباعة عشرين عنصراً من المصفوفة ربما نصادف المزيد من

الفضلات كما في شكل (٧ - ١٧) .

Running

0  
0  
0  
0  
0  
0  
0  
0  
0  
0

-32768  
358  
5632  
12336  
12336  
12336  
12336  
12336  
48  
358  
5682  
0

مخلفات  
بيانات

>

شكل (٧ - ١٧)

لذلك يجب « تصفير » جميع المتغيرات قبل التعامل معها حتى لا تتسبب في الحصول على نتائج خاطئة .

(٧ - ٦) معالجة المصفوفات ذات البعدين :

يمثل شكل (٧ - ١٨) مصفوفة ذات بعدين مكونة من ثمانية صفوف وعشرة أعمدة ، فإننا عادة نقرأ المصفوفة صفّاً صفّاً من خلال حلقة تكرارية كالآتي :

**for row := 1 to 8do**

...

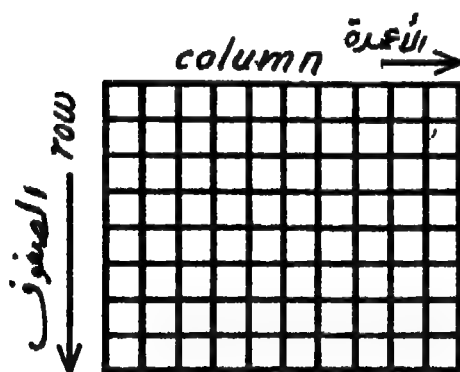
(عملية قراءة أو معالجة الصف) ...

ولكن عملية قراءة أو معالجة صف واحد تتضمن معالجة الأعمدة العشرة التي تساهم في تكوين كل صف . لذلك فإن عملية القراءة للصف الواحد سوف تكون عبارة عن حلقة تكرارية أخرى بداخل الحلقة الأولى كالآتي :

**for column := 1 to 10 do**

...

(عملية قراءة أو معالجة لكل عنصر)



شكل (٧ - ١٨)



ولنر هذا المثال :

مثال (٧ — ٨) تعداد السكان :

لو اعتبرنا أن كل مربع صغير في شكل (٧ — ١٨) من المربعات الثمانية يمثل منطقة من المناطق السكنية على خريطة المدينة ، فيمكن اعتبار أن الشكل كله يصور لنا تعداد السكان في مدينة ما .

فإذا أردنا الإعلان عن هذه المصفوفة فلنمنحها اسماً مثل popmap بمعنى خريطة التعداد .

وللإعلان عن المصفوفة نكتب السطر التالي :

**var popmap : array [1..8,1..10] of integer;**

ويصبح بذلك المتغير الدليل العام هو :

**popmap[row, column]**

حيث row هو رقم الصف .

column هو رقم العمود .

والبرنامج التالي يستخدم هذه المصفوفة في حساب التعداد الكلي للسكان (total) ويطبّع هذا التعداد علاوة على تعداد كل منطقة من المناطق . شكل (٧ — ١٩) ويبدأ البرنامج بقراءة المصفوفة .. وهذا يستلزم منا كتابة ثمانية رقماً مختلفاً إذا كنا سوف نستخدم لوحة الأزرار .. ونلاحظ في تنفيذ البرنامج أن الأرقام المدخلة لا تحتوى أية فواصل فيما بينها ولكن كل رقم مع ذلك يعقبه الضغط على زر الإدخال (ENTER) . وقد استخدمنا أرقاماً صغيرة بين الصفر والتسعة لتمثيل عدد السكان بالآلاف في كل منطقة لتسهيل عملية إدخال البيانات .

أما الجزء الثاني من البرنامج فهو تجميع العناصر كلها في المتغير : total  
وطباعته .

والجزء الأخير هو طباعة عناصر المصفوفة كلها لتمثيل عدد السكان في كل  
منطقة .

★ ★ ★

تعداد السكان

```
program population(input,output);
VAR popmap: array[1..8,1..10] of integer;
    row,column,total: integer;
```

BEGIN

```
for row :=1 to 8 do
  for column :=1 to 10 do
    read(popmap[row,column]);
```

قراءة المصفوفة

```
total:=0;
for row :=1 to 8 do
  for column :=1 to 10 do
    total:=total+popmap[row,column];
```

تجميع عناصر المصفوفة

```
writeln;
writeln('total population is ',total);
writeln('populations of individual zones are:');
```

طباعة التعداد الكلي

```
for row :=1 to 8 do
  BEGIN
    writeln;
    for column :=1 to 10 do
      write(popmap[row,column]);
    END;
```

طباعة عناصر المصفوفة

تنفيذ البرنامج:

Running  
98999766578694319122233465673267889654332222111899858654323335545454566765676  
total population is 399  
populations of individual zones are:

لتر إدخال ٨٠ عنصراً

التعداد الكلي

عناصر المصفوفة

```
9899976657
8694319122
2233465673
2267889654
3322222111
8998586543
2333554545
4566765676
>
```

## مثال (٧ - ٩) .. تحليل تعداد السكان :

في هذا البرنامج يتم قراءة عناصر المصفوفة الممثلة لتعداد السكان كما في المثال السابق ثم يقوم البرنامج بإيجاد المناطق التي يقل تعداد السكان فيها عن نصف متوسط تعداد السكان في المناطق المحيطة بها من الجهات الأربع (الشمال والجنوب والشرق والغرب) .

وحساب المتوسط يتم بإضافة تعداد السكان في الأربعة مربعات المحيطة بكل مربع وقسمة الناتج على أربعة . فإذا كانت إحداثيات أى مربع عام هي :  
row, col فإن الإحداثيات الآتية تمثل المربعات الأربعة المحيطة :

- المربع الشرق row, col+1
- المربع الغرب row, col-1
- المربع الشمال row-1, col
- المربع الجنوب row+1, col

```

program popmap3(input,output);

var popmap : array [1..8,1..10] of integer;
    row, col : integer;  average : real;

begin
    for row := 1 to 8 do
        for col := 1 to 10 do
            read(popmap[row,col]);

            for row := 2 to 7 do
                for col := 2 to 9 do
                    begin
                        average := (popmap[row-1,col]
                                    +popmap[row+1,col]
                                    +popmap[row,col-1]
                                    +popmap[row,col+1])/4;
                        if popmap[row,col] < average/2
                        then writeln('the region ', row, col,
                                    ' is underpopulated')
                    end
                end
            end
        end
    end.

```

المتوسط

شكل (٧ - ٢٠)

## (٧ - ٧) ملاحظات على المتغيرات الدلالية :

علمنا من قبل أن دليل المتغير في مصفوفة ما يأخذ قيمة تبدأ من الواحد وتنتهي بالعدد الذي يعبر عن بُعد المصفوفة . كما علمنا أنه من الجائز أن يبدأ الدليل من قيمة أخرى خلاف الواحد . وبصفة عامة يمكن أن يكون الدليل ثابتاً أو ثابتاً مسمى .

ولتر هذا المثال حيث يعرض علينا حالة إحدى الشركات التي تستخدم مسلسلًا رقميًا لتشفير السلع التي تبيعها ، وتقع الأرقام المسلسلة في النطاق من 3001 إلى 3563 . والبرنامج يقرأ قائمة الأسعار لهذه الأصناف ، ثم يقرأ الكمية المطلوبة من كل صنف ويحسب السعر الكلي للطلبية . وتستخدم علامة النجمة \* لإنهاء إدخال البيانات .

## مثال (٧ - ١٠) حساب المبيعات :

```
program sale(input,output);
const minref = 3001; maxref = 3563;
var priceof : array [minref..maxref] of real;
    ref, nextref, quantity : integer;
    totalprice : real; nextch : char;
begin
    for nextref := minref to maxref do
        read(priceof[nextref]);
    totalprice := 0;
    repeat
        read(ref,quantity);
        totalprice := totalprice + priceof[ref]*quantity;
        read(nextch);
    until nextch = '*';
    writeln('total cost of order is ', totalprice);
end.
```

مبيعات

الإعلان باستخدام الثوابت المسماة

قراءة الأسعار

حساب سعر الطلبية

## شكل (٧ - ٢١)

مثال (٧ - ١١) معدل تكرار الحروف في نص :

هل يمكن استخدام اللبئات كأدلة للمصفوفة ؟

في هذا المثال نعرض تطبيقاً لاستخدام الحروف الأبجدية (اللبئات) كأدلة لمصفوفة من الأعداد الصحيحة . ويقوم البرنامج بحساب معدل تكرار كل حرف في قطعة من نص تنتهي بالعلامة (\*) ويتم التعبير عن معدل التكرار بتوقيعه كرسوم بياني .

ويبدأ البرنامج بشحن المتغيرات الدليلية بالقيمة « صفر » ثم يقرأ لبنة ويختبر ما إذا كانت تنتمي إلى الفئة ["a".."z"] أي فئة الحروف الأبجدية ، وفي هذه الحالة يضيف واحداً إلى عنصر المصفوفة الذي يمثل عدداً هذه اللبنة . فإذا كان المتغير الدليلي lettercount["a"] يحتوي على العدد 5 كان معنى ذلك أن الحرف 'a' تكرر خمس مرات وهكذا بالنسبة لبقية الحروف .

```

program freqcount(input,output);
var lettercount : array ['a'..'z'] of integer;
    letter, character : char; col : integer;
begin
    for letter := 'a' to 'z' do
        lettercount[letter] := 0;

    read(character);
    repeat
        if character in ['a'..'z'] then
            lettercount[character] := lettercount[character] + 1;
        read(character);
    until character = '*';

    for letter := 'a' to 'z' do
    begin
        writeln;
        for col := 1 to lettercount[letter] do
            write('*')
        end
    end
end.

```

*اللبئات كأدلة*

*اللبئات كعداد*

*حساب معدل تكرار حرف*

*رسم المستوجرام*

شكل (٧ - ٢٢)

## والتغيرات المستخدمة هي :

- lettercount مصفوفة أعداد صحيحة ذات أدلة من الحروف الأبجدية تستخدم لحساب معدل تكرار كل حرف .
- letter متغير لبنة يحتوى على أحد الحروف الأبجدية ويستخدم كدليل للمصفوفة .
- character متغير لبنة يحتوى على أحد لبنات النص تحت المعالجة .
- col متغير صحيح يمثل العمود الرأسى (الخانة) المستخدم فى توقيع الرسم .

وهذا البرنامج لا يعدّ ذات قيمة عملية إلا إذا تمت قراءة النص من ملف .

## (٧ - ٨) الأنماط المبكرة Type :

عندما تحدثنا عن الأنماط من قبل عرضنا أنواعاً أربعة من الأنماط وهى :  
الصحيحة والحقيقية والمنطقية وأنماط اللبنات .

Integer

Real

Boolean

Char

ومع ذلك ففى بعض الأحيان قد نحتاج لتمثيل أنواع من البيانات لا تندرج تماماً تحت هذه الأنماط . وعلى سبيل المثال إذا أردنا تمثيل عدد الدقائق الزمنية الموجودة فى كسر الساعة (وهى تقع فى النطاق من 1 إلى 59) فمن المنتظر فى حدود ما عرفناه حتى الآن أن نعلن عن متغير مثل minute من النوع الصحيح integer .

وعلى المبرمج في هذه الحالة أن يمد برنامجه بالوسائل اللازمة لحمايته من البيانات غير الصحيحة مثل العدد 82 أو العدد السالب 28 .

هناك وسيلة أكثر سهولة تحققها لنا لغة باسكال باستخدام فئة « جزئية » من الأعداد الصحيحة فقط .

(٧ — ٨ — ١) أنماط الفئات الجزئية (subranges) :

يمكننا أن نُعرّف الفئة الجزئية للأعداد الصحيحة التي تمثل الدقائق (كسور الساعة) كالآتي :

**VAR minutes : 1..59;**

معنى ذلك أن المتغير minutes لا يجوز أن يحتوي على أى قيمة صحيحة خارج نطاق الأعداد من 1 إلى 59 ولذلك يطلق عليه الاسم الفئة الجزئية (subrange) ، حيث أنه يعتبر فئة جزئية من فئة الأعداد الصحيحة كلها .

ويجوز الإعلان عن أية فئات جزئية مادامت فئة ذات عدد محدود من العناصر . (الأرقام الحقيقية لا تخضع لهذا التعريف لأن الأرقام التي تنحصر بين الرقمين 5 ، 7 مثلاً لا يمكن حصرها) لكنه من الجائز إنشاء فئة جزئية من الحرفيات كالمثال الآتي :

**VAR numerics : '0'..'9';**

ولاستخدام الفئات الجزئية عدة مميزات منها :

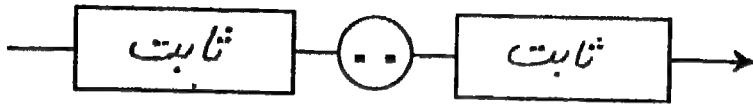
(١) منع البيانات غير الصحيحة من المعالجة بواسطة البرنامج حيث يقوم البرنامج بإرسال رسالة الخطأ المناسبة .

(٢) تقليل حجم الذاكرة المخصصة لتمثيل البيانات بالأرقام الثنائية .

والشكل (٧ — ٢٣) يوضح قاعدة تكوين نمط الفئات الجزئية



بالرسم .



شكل (٧ - ٢٣)

(٧-٨-٢) أنماط القوائم Enumeration types :

في الكثير من لغات الكمبيوتر قد نحتاج إلى تمثيل أيام الأسبوع بالأرقام وذلك بإعطاء كود لكل يوم مثل :

**monday = 0 .... sunday = 6**

وهذا يقابله في لغة باسكال إعلان المتغير dayofweek كصفة جزئية من 0 إلى 6 كالآتي :

**VAR dayofweek : 0..6**

وبذلك يمكن أن يتضمن البرنامج عبارة مثل :

**IF (dayofweek = 5) OR (dayofweek = 6)  
THEN**

بمعنى إذا كان اليوم هو يوم السبت (رقم 5) أو يوم الأحد (رقم 6).....  
كما يجوز إعلان أيام الأسبوع كتوابت مسماه كالآتي :

```
CONST mon = 0; tues = 1; wed = 2; thurs = 3;
    fri = 4; sat = 5; sun = 6;
```

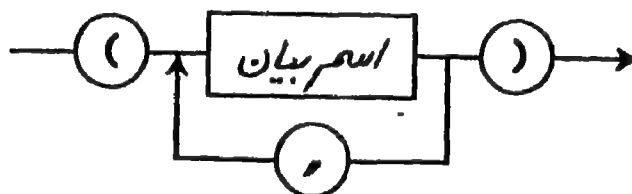
وفي هذه الحالة فإننا يمكن أن نستخدم الحروف بدلاً من الأرقام في التعامل مع الأيام خلال البرنامج إذ يمكننا كتابة العبارة الشرطية السابقة بصورة أفضل وأكثر وضوحاً للقارئ :

```
IF (dayofweek = sat) OR (dayofweek = sun)
    THEN ...'
```

ومع ذلك فكل هذه المحاولات تحتاج جهداً في البرمجة ، ويمكننا الاستغناء عنها جميعاً باستخدام نمط القائمة كآآتي :

```
VAR dayofweek : (mon,tues,wed,thurs,fri,sat,sun);
```

وقاعدة تكوين نمط القائمة موضحاً بالرسم في شكل (٧-٢٤) .



شكل (٧ - ٢٤)

هذا التعريف يتضمن تلقائياً تمثيل الأيام بأرقام كودية من 0 إلى 6 كخاصية من خصائص اللغة دون أن يشعر المبرمج ودون أن يبذل جهداً في ذلك . وفي هذه الحالة يمكننا استخدام نفس العبارة الشرطية السابقة باستخدام الثوابت

الواردة في الإعلان :

**IF (dayofweek = sat) OR (dayofweek = sun)**  
**THEN...**

ومن البديهي أنه لا يجوز استخدام الثوابت العددية للتخصيص للمتغير dayofweek في هذه الحالة ؛ فالعبارة الآتية تصبح عبارة خاطئة :

**IF (dayofweek = 5) OR (dayof week = 6)**  
**THEN**

وذلك لأن الأعداد 5 ، 6 ليست متضمنة في القائمة المعلن عنها صراحة .  
كذلك لا يجوز إجراء أية عمليات حسابية على الثوابت sun ، sat ... إلى آخره . فلايجاد اسم اليوم المعبر عن « باكر » أو « أمس » لا يجوز الجمع والطرح بل تستخدم لذلك الدوال succ ، pred كالمثال الآتي :

```
VAR yesterday, today, tomorrow:
    (mon,tues,wed,thurs,fri,sat,sun);
:

IF today = sun THEN tomorrow:= mon
ELSE tomorrow:= succ (today)
:
IF today = mon THEN yesterday:= sun
ELSE yesterday:= pred (today)
```

(باكر) اليوم التالي ←      ← اليوم السابق (أمس)

شكل (٧ - ٢٥)

ويمكنك التأكد من أن الثوابت sun ، sat ، ... تناظر أعداداً بطبيع الكود المناظر لها باستخدام الدالة Ord كالآتي :

ord (mon) ← تعطى القيمة 0  
ord (sum) ← تعطى القيمة 6

لا شك أن هذه الخاصية تسهل على المبرمج مراجعة برنامجه حيث يصبح مكتوباً بلغة مفهومة كما أنها تسهل على القارئ متابعة ما يقرأ من عبارات باسكال . ولتر هذه العبارة الجميلة المعبرة عن قائمة من ألوان الطبيعة :

**VAR colour : (red,green,blue,white);**

إنها تعبر عن نفسها .

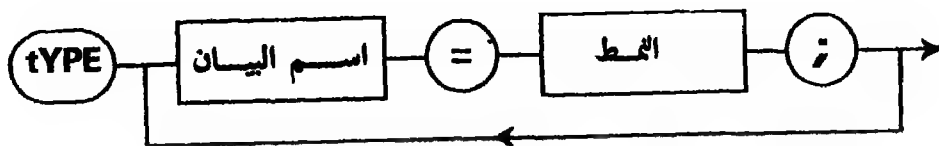
**type** (٣ — ٨ — ٧) العبارة :

في الواقع أن الفئات الجزئية والقوائم ما هي إلا أنماط مبتكرة يمكن استخدام أسمائها تماماً كما تستخدم الأنماط القياسية والصورة الكاملة للأنماط المبتكرة تكتمل مع استخدام العبارة TYPE التي تعني تعريف نمط جديد والمثال التالي يوضح استخدامها للتعريف بأيام الأسبوع وعدد الثواني كأنماط جديدة تحت الإسمين **day** ، **minutetype** شكل (٧ — ٢٦) .

فئة الدقائق ← **TYPE minutetype = 1...59;**  
**day = (mon,tues,wed,thurs,fri,sat,sun);**  
**VAR minutes : minutetype;**  
**today,tomorrow: day;**  
قائمة الأيام ↑

شكل (٧ — ٢٦)

وقاعدة استخدام العبارة **type** موضحة في شكل (٧ — ٢٧) بالرسم :



شكل (٧ - ٢٧)

● الثابت القياسي maxint :

لكل طراز من طرازات لغة باسكال ثابت قياسي يحمل الاسم maxint (اختصار maximum integer) بمعنى أكبر عدد صحيح موجب يمكن تمثيله في الكمبيوتر . ولكي تكتشف قيمة هذا العدد في جهازك الخاص اكتب هذا البرنامج :

```
begin
writeln('maxint is',maxint)
end.
```

البرنامج

```
Running
maxint is 32767
```

التنفيذ على IBM

>

شكل (٧ - ٢٨)

ومع الكمبيوتر الشخصي IBM يتوقع أن يكون هذا الرقم 32767 . ولهذا  
الثابت القياسى منافع مختلفة فى تعريف الأنماط المبتكرة . مثل تعريف نمط  
الأعداد غير السالبة (أى فئة الأعداد الموجبة والصفر) ، وتعريف نمط الأعداد  
الطبيعية (أى فئة الأعداد الموجبة) . والمثال التالى يوضح هذه التعريفات  
باستخدام الثابت القياسى maxint (شكل ٧ - ٢٩) :

```

TYPE nonnegative = 0..maxint;
      natural      = 1..maxint;

```

فئة الأعداد الطبيعية ← natural  
فئة الأعداد غير السالبة ← nonnegative

شكل (٧ - ٢٩)

#### ● الفئات الجزئية من القوائم :

إن الإعلان عن نمط من أنماط القوائم يتيح استخدام أسماء البيانات الواردة فى  
القائمة ككوابت . وبالتالى يجوز استخدامها فى الإعلان عن فئة جزئية جديدة .

والمثال التالى يقدم النمط day الذى يحتوى على قائمة بأيام الأسبوع ثم  
يستخدم القائمة ذاتها فى التعريف بالفئة الجزئية المعبرة عن أيام العمل (من  
الأحد إلى الخميس) بفرض أن أيام العطلات هى الجمعة والسبت ، وكذلك  
بالفئة الجزئية لأيام العطلات .

```

type day = (mon,tue,wed,thurs,fri,sat,sun);
workday = sun..thurs; ← أيام العمل
holiday = fri..sat;    ← أيام العطلات

```

شكل (٧ - ٣٠)

● ● ولعل المثال الأخير يوضح الفارق الأساسى فى المعنى بين القائمة والفئة الجزئية فكلاهما يعبر عن فئة ولكن الفئة الجزئية هى فئة متصلة أما القائمة فهى فئة غير متصلة . وعلى سبيل المثال فإن لاعبى كرة القدم ينتمون إلى فئة الفريق ، فإذا ميزناهم بأسمائهم فإنهم يعتبرون قائمة وإذا ميزناهم بأرقام مسلسلته فإنهم يصبحون فئة جزئية ١ .

● النمط البوليائى **type boolean** :

يمكن تعريف النمط البوليائى القياسى بالصورة الآتية باعتباره نمط مبتكر :

**TYPE boolean = (false,true);**

وبذلك فإن الدالة **ord(false)** تعطى القيمة صفراً .

والدالة **ord(true)** تعطى القيمة ١ .

ولذلك فانمط البوليائى ينتمى إلى أنماط القوائم .

● الموجز :

يمكن الإعلان عن الفئات الجزئية من الأنماط الآتية :

١ — الصحيح integer

٢ — اللبنة char

٣ — البوليائى (ولكن لا قيمة له) boolean

كما يمكن الإعلان عن أنماط القوائم من أى نوع ما عدا النمط الصحيح real .

(٧ — ٩) أدلة المصفوفات :

بعد أن عرضنا موضوع الفئات الجزئية والقوائم لعلنا نرى العلاقة بين المصفوفة وبين الفئات والقوائم واضحة الآن .

فعندما نعلن مصفوفة صحيحة مثل :

**array [1..9] of integer**

فإن أدلة هذه المصفوفة ما هي إلا الفئة الجزئية من الأعداد الصحيحة من 1 إلى 9 . كما رأينا أيضاً أن أدلة المصفوفة يجوز أن تكون فئة جزئية من اللبنات  
مثل :

**array ['a'..'z'] of integers**

وأدلة المصفوفات بصفة عامة يجوز أن تكون من أحد الأنواع الآتية :

- ١ — فئات جزئية من الأعداد الصحيحة .
- ٢ — فئات جزئية من اللبنات .
- ٣ — فئات جزئية من الثوابت المنطقية .
- ٤ — القوائم بأنواعها .

مثال (٧ — ١٠) :

وفي المثال التالي نرى أن دليل المصفوفة عبارة عن قائمة تحتوي على شهور السنة .



```

TYPE monthype = ( jan, feb, mar, apr,
                  may, jun, jul, aug,
                  sep, oct, nov, dec );
VAR  rainfall : ARRAY [ monthype ] OF real;
      thismonth,
      finalmonth : monthype;
      total : real;
BEGIN
  finalmonth := aug;
  total := 0.0;
  FOR thismonth := jan TO finalmonth DO
    total := total + rainfall [ thismonth ];
  END.

```

أدلة المصفوفة

شكل (٧ - ٣١)

مثال (٧ - ١١) :

أما المثال التالي فهو يستخدم أدلة كثيفة جزئية من الأعداد الصحيحة الممثلة للسنوات من 1919 حتى 1938 ، أما نمط المصفوفة نفسها فهو من الأنماط المبتكرة وهو عبارة عن فئة الأعداد غير السالبة (posint) أى التى تبدأ من الصفر وحتى maxint .

```

TYPE posint = 0..maxint;
      betweenwars = 1919..1938;
VAR  _unemployed : array [betweenwars] OF posint;
      year       : betweenwars;
      posvar      : posint;
      intvar      : integer;

```

شكل (٧ - ٣٢)

في هذا المثال يمكن الإشارة إلى أحد المتغيرات الدليلية بالمصفوفة كآلاتي :

**unemployed[year]**

حيث يعبر المتغير year عن أحد السنوات في الفئة [1919..1938] .

● كما يجوز الإشارة إلى المتغير الدليلي للمصفوفة باستخدام الدليل (posvar) الذي تم تعريفه بأنه عدد صحيح موجب .

**unemployed[posvar]**

ولكن استخدام مثل هذا الدليل سيتطلب من البرنامج إجراء مراجعة أثناء التنفيذ للتأكد من أن قيمة هذا المتغير تقع في نطاق الفئة [1919.1938] ، فهذه الفئة تمثل حدود الدليل المستخدم للمصفوفة .

وكذلك الحال إذا استخدمنا الدليل (intvar) الذي تم تعريفه كعدد صحيح .

والاستخدام الأمثل هو استخدام الدليل الأول (year) .

مثال (٧ - ١٢) :

وهذا مثال لمصفوفة تحتوي على أربعة مواسم تجارية تم الإعلان عنها كقائمة من أربعة عناصر (quarter) وأطلق عليها الاسم figures كنمط مبتكر .

```
TYPE quarter = (first, second, third, fourth);
figures = ARRAY [quarter] OF integer;
period = 1976..1980;
VAR profits, losses, turnover : figures;
    fiveyearprofits : ARRAY [period] OF figures;
    current : quarter;
```

*(مصفوفة ذات بعد واحد)*  
*(مصفوفة ذات بعدين)*

شكل (٧ - ٣٣)

ونلاحظ أنه تم تعريف متغيرات ثلاثة كمصفوفات من نفس النمط . هذه المتغيرات هي :

profits	الأرباح
losses	الخسائر
turnover	جملة المبيعات

فإذا أردنا تمثيل أرباح الموسم التجارى ، فيمكننا الإشارة إليه بالمتغير الدليلي :

**profits [second]**

وجملة مبيعات الموسم الثالث هي :

**turnover [third]**

أما خسائر الموسم الحالى (current) فهي :

**Losses [current]**

●● وإذا دققنا النظر للإعلان عن المتغير fiveyearprofits لوجدنا أنه « مصفوفة مصفوفة » أى مصفوفة ذات بعدين . وهذه وجهة نظر أخرى للمصفوفة ذات البعدين حيث يمكن اعتبارها مكونة من مصفوفات ذات بعد واحد وهى الصفوف (أو الأعمدة) . لذلك يمكن الإشارة إلى أحد عناصر هذه المصفوفة كالآتى :

**fiveyearprofits [1977,first]**

وهو يمثل أرباح الموسم الأول من ١٩٧٧ .  
كما يجوز التعبير عن نفس المتغير كالآتى :

**fiveyearprofits [1977] [first]**

## (٧ — ١٠) المصفوفات المُحزّمة *Packed arrays* :

هذه نوعية من المصفوفات صممت لتشغل حيزاً أقل من سعة الذاكرة لكنها لا تتوفر بكل طرازات اللغة .

ويتم الإعلان عن هذه النوعية من المصفوفات بإضافة كلمة packed قبل كلمة array في الإعلان عن المصفوفة .

وهذا الإعلان يتم فيه التعريف بمصفوفتين A1 ، A2 الأولى من النوع المحزّم والثانية من النوع العادى وكلاهما من النمط الحقيقى :

```
VAR      A1 : PACKED ARRAY [1..4] OF real;  
        A2 :  ARRAY [1..4] OF real;
```

وفائدة المصفوفات المحزّمة تظهر في التعامل مع الحرفيات حيث أن لغة باسكال القياسية قد عرّفت الحرفيات (المتغيرات الحرفية) بأنها مصفوفة لبنات محزّمة كالآتى :

**PACKED ARRAY [1..n] OF char;**

حيث n هو عدد اللبّات في الحرفى المقصود .

ومن خصائص عبارة الطباعة (write أو writeln) أنها تطبع هذه الحرفيات (راجع الباب الثالث) .

ويتم التعامل مع مصفوفة الحرفيات المحزّمة بنفس الأسلوب الذى يتم به التعامل مع المصفوفات .

فيمكن التخصيص لمتغير حرفى كالآتى (بنفس عدد اللبّات) :

```
VAR name : PACKED ARRAY [1..10] OF char;  
name := 'pascal      ';
```

كما يمكن مقارنة الحرفيات باستخدام المؤثرات العلاقية كالآتي :

```
IF name = 'pascal
  THEN...
IF name < 'pascal
  THEN...
```

وتتم المقارنات بين الحرفيات لبنة بلبنة من اليسار إلى اليمين وفقاً للكوند آسكى (ASCII) أو الكوند EBCDIC فى بعض الطرازات .

مثال (٧ - ١٣) على الكومبيوتر آى . بى . إم :

يوضح شكل (٧ - ٣٤) برنامجاً يستخدم مصفوفة اللبئات المحزّمة المكونة من ١٢ عنصراً .

ويبدأ البرنامج بقراءة الاسم من لوحة الأزرار ثم يقوم البرنامج باختبار الاسم الذى تمت قراءته إذا كان هو الاسم « عبد الفتاح » وفى حالة إذا ما تحقق الشرط فإنه يعطى الرسالة :

**ok...name found**

ولاً فإنه يطبع الاسم المدخل كما هو .

ونلاحظ مع العبارة read أنه يمكن إدخال عدد من اللبئات أقل من أبعاد المصفوفة . وهذا لا يجوز عند استخدام التخصيص .

```

program demo(input,output);
var name:packed array[1..12] of char;

begin
  readln(name);
  if name = abdel-fattah then
    writeln('ok..name found.'):
    else writeln(name)
  end.

```

البرنامج

```

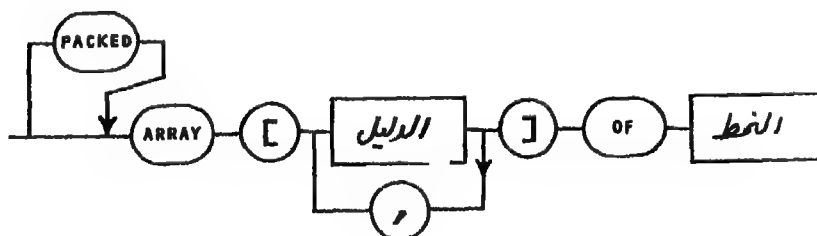
Running
ahmad      ← الاسم المدخل "أحمد"
ahmad      ← لم يتحقق الشرط
:
Running
abdel-fattah → الاسم المدخل "عبد الفتاح"
ok..name found. ← تحقق الشرط
:

```

التنفيذ

شكل (٧ - ٣٤)

والشكل الآتي يوضح قاعدة تكوين المصفوفات والمصفوفات المحزّمة بالرسم :



شكل (٧ - ٣٥)

## (٧ - ١١) الحرفيات في طرازات باسكال *string* :

كما نرى أن الحرفيات في اللغة القياسية كانت مقيدة ! ولكن بعض الطرازات قد أدخلت الحرفيات بمعناها الدقيق كما في اللغات الأخرى .  
وقد تم ذلك بإدخال غمط جديد هو الغمط الحرفي *string* بحيث يمكننا أن نعلن عن متغير حرفي بالصورة الآتية :

**VAR name : string;**

وهذا المتغير الحرفي يسمح باستقبال ثوابت حرفية يصل طولها إلى ثمانين لبنة (في الطراز UCSD) .  
وعلاوة على ذلك فإنه يمكن الإعلان عن أى طول متوقع للحرفي يزيد عن الثمانين كالآتي :

**string [n]**

حيث n : عدد صحيح لا تزيد قيمته عن ٢٥٥ .

●● ملاحظة : هذه الصيغة المسموح بها في الكومبيوتر IBM والطرازات المتوافقة معه أما الصيغة الأولى فغير متوفرة) .  
ويتم الإعلان كالمثال الآتي :

**VAR carreg : string [7];**

**printline : string [132];**

وعند التخصيص للمتغير الحرفي يجوز أن نخصص له ثابتاً حرفياً بطول أقل أو يساوى الطول المعلن عنه كالآتي :

**carreg := 'abc123x'**

أو

**carreg := 'aa1'**

كما يجوز فهرسة الحرفي (على اعتبار أنه مصفوفة) وفي هذه الحالة يمكن الإشارة إلى أى لبنة فيه كأنها متغير دليل له ترتيب معين كالمثال الآتي :

```
VAR name : string;
    ch    : char;
name := 'kenneth';
ch := name [6];
```

الحرف السادس

### شكل (٧ - ٣٦)

فالتغير اللبنة ch قد أصبح محتوياً على الحرف t بعد عملية التخصيص حيث أن الحرف t هو الحرف السادس في مصفوفة المتغير الحرفي name .

### (٧ - ١٢) دوال الحرفيات : *string functions*

إن الطرازات التي يتوفر بها النمط string تمدنا في نفس الوقت بكل الوسائل اللازمة لمعالجة الحرفيات كما في لغة بيسك .

وهذه الوسائل عبارة عن دوال وبرامج متوفرة ضمن مترجم اللغة تماماً كاللوال القياسية .

#### (١) قياس طول الكلمة *length* :

صيغة هذه الدالة :

#### **Length (string)**

وهي تستخدم لإيجاد عدد لبنات الحرفي (string) الذي بين القوسين ، ونتيجة هذه الدالة عدد صحيح يجوز تخصيصه لأي متغير صحيح .



مثال (٧ - ١٤) :

البرنامج التالي يستقبل اسماً ما name ويخزّنه في متغير حرفي يتسع لعشرين لبنة ثم يقيس طول الاسم (عدد لبناته) ويخصّصه للمتغير الصحيح l.

```
program length(input,output);
var name:string[20];
    l:integer;

begin

readln(name);
l:=length(name);
writeln(name,' of length ',l);

end.
```

البرنامج

قياس طول الكلمة

التنفيذ

```
Running
ibrahim soliman ← الاسم المدخل
ibrahim soliman of length 15 ←
>                                طول الاسم
```

```
Running
sally o. al-husseiny ← الاسم المدخل
sally o. al-husseiny of length 20 ←
>                                طول الاسم
```

شكل (٧ - ٣٨)

## (٢) ترتيب لبنة داخل حرفي POS :

تعطى هذه الدالة عدداً صحيحاً يعبر عن ترتيب لبنة معنية أو أكثر (حرفي جزئي) بداخل حرفي وإذا كان الحرفي الجزئي متكرراً أكثر من مرة بداخل هذا الحرفي فإن هذه الدالة تعطى ترتيباً أول حدوث للحرفي الجزئي .

وصيغة الدالة هي :

**pos (substring, string)**

حيث :

substring	الحرفي الجزئي المطلوب إيجاد موضعه .
string	الحرفي المعلوم .

## مثال (٧ - ١٥) :

هذه الشريحة من البرنامج تبحث عن ترتيب الكلمة and في المتغير الحرفي authors وتخصص النتيجة للمتغير الصحيح a وعند طباعة قيمة المتغير a سوف تكون 8 لأن بداية الكلمة and تقع في الخانة الثامنة .

```
VAR l : integer;
    authors : string [20];
authors := 'jensen and wirth';
l := pos ('and', authors);
```

## شكل (٧ - ٣٩)

وإذا فشلت عملية البحث عن الحرفي الجزئي فإن الدالة تعطى القيمة صفراً . ولنر هذا المثال :

(مثال ٧ - ١٦) :

يوضح الشكل التالى برنامجاً يستقبل اسماً ويخزنه فى الحرفى name ثم يستقبل لبنة واحدة ليبحث عن وجودها فى الحرفى name ثم يطبع على الشاشة موضع هذه اللبنة من الاسم . فإذا كانت اللبنة غير موجودة أعطى القيمة صفراً .

```
program position(input,output);
var name:string[255];
    l:integer;
    letter:char;

begin
writeln('please enter a name...');
readln(name);
writeln('what is character you're looking for?');
readln(letter);
l:=pos(letter,name);
writeln(letter,' is in the position no:',l);

end.
```

```
>

Running
please enter a name...
mahmood abbas
what is character you're looking for?
b
b is in the position no:10
>
```

```
Running
please enter a name...
aly abdel-gawwad
what is character you're looking for?
z
z is in the position no:0
>
```

شكل (٧ - ٤٠)

وبالطبع يمكن تعديل البرنامج بحيث يخبرنا برسالة مفهومة : إذا ما كانت  
اللينة المراد إيجاد موضعها موجودة أصلاً في الحرفي أم لا .  
وهذا التعديل نراه في البرنامج شكل (٧ — ٤١)

\* \* \*

```

program position2(input,output);
var name:string[255];
    l:integer;
    letter:char;

```

البرنامج

```

begin
writeln('please enter a name...');
readln(name);
writeln('what is character you're looking for?');
readln(letter);
l:=pos(letter,name);
if l=0 then
writeln('the character doesn't exist')
else
writeln('letter, ' is in the position no:',l);

end.

```

البحث عن حرف في كلمة

التعديل المطلوب

التنفيذ

```

please enter a name...
MEDHAT ASSAR
what is character you're looking for?
s
the character doesn't exist

```

الاسم

الحرف المراد البحث عنه

النتيجة... الحرف غير موجود

```

Running
please enter a name...
medhat assar
what is character you're looking for?
s
s is in the position no:9

```

الاسم

الحرف المراد البحث عنه

النتيجة

```

Running
please enter a name...
abdel-salam shamah
what is character you're looking for?
r
the character doesn't exist

```

الاسم

الحرف

غير موجود

فإن بين هذين التنفيذين

شكل (٧ - ٤١)

(٣) وصل الكلمات concat :

تقوم هذه الدالة بوصل الحرفيات بعضها ببعض وصيغتها كالآتي :

**concat (string, string, ..string)**

ونتيجة الدالة هي عبارة عن حرفي مكون من الحرفيات الموجودة بداخل القوس ، والمفصولة عن بعضها البعض بفاصلة .

مثال (٧ - ١٧) :

يوضح شكل (٧ - ٤٢) التنفيذ المتوقع للبرنامج الذي سنعرضه حيث يستقبل ثلاثة أسماء هي : اسم الشخص واسم الأب (أو الاسم الأوسط) واسم العائلة ثم يطبع الاسم كاملاً مع اختصار الاسم الأوسط إلى حرف واحد يعقبه نقطة .

Running  
please enter first name...  
SALLY ← الاسم الأول  
please enter the middle name...  
OSSAMA ← اسم الأب  
please enter the family name  
ABOLROUS ← اسم العائلة  
the complete name is : SALLY O. ABOLROUS

التنفيذ المتوقع للبرنامج

> البرنامج يطبع الاسم كاملاً مع اختصار اسم الأب

شكل (٧ - ٤٢)

أما البرنامج فيمكن إنشاؤه كما هو موضح في شكل (٧ - ٤٣) حيث يستقبل الأسماء الثلاثة a,b,c وكلها متغيرات حرفية تمثل الاسم واسم الأب

واسم العائلة بالترتيب .

ثم يلي ذلك اختصار الاسم الأوسط إلى حرف واحد وذلك بتخصيص الحرف الأول منه **b[1]** للمتغير **a** .

يلي ذلك استخدام دالة الوصل لكي تصل بين الحرفيات الثلاثة a, b, c علاوة على المسافات الخالية بين الأسماء .

البرنامج ١

```
program completename(input,output);
```

```
var a,b,c,d:string[20];
```

```
l:string[1];
```

```
begin
```

```
writeln('please enter first name...');
```

```
readln(a);
```

```
writeln('please enter the middle name...');
```

```
readln(b);
```

```
writeln('please enter the family name');
```

```
readln(c);
```

```
l:=b[1];
```

```
d:=concat(a,' ',l,' ',c);
```

```
writeln('the complete name is :',d);
```

```
end.
```

وصل الحرفيات

اختصار الاسم الأوسط

الوصل

شكل (٧ - ٤٣)

أما البرنامج شكل (٧ — ٤٤) فهو يعرض نفس البرنامج مع الاستغناء عن المتغير الوسيط والاستعاضة عنه بتعديل سعة المتغير (b) المخصص لاستقبال الاسم الأوسط لتكون لبنة واحدة فقط أى :

**b:string[1]**

في هذه الحالة ... عندما يُستقبل الاسم الأوسط أثناء تشغيل البرنامج لن يُخترن منه في المتغير (b) سوى حرف واحد فقط .

```
program completename(input,output);
var a,c,d:string[20];
    b:string[1];
begin
    writeln('please enter first name...');
    readln(a);
    writeln('please enter the middle name...');
    readln(b);
    writeln('please enter the family name');
    readln(c);

    d:=concat(a,' ',b,'.',',',c);
    writeln('the complete name is :',d);
end.
```

(البرنامج)

متغير يتسع للبنة واحدة

وصل الحرفيات

الوصل

شكل (٧ — ٤٤)

(٤) نسخ جزء من كلمة copy :

تقوم هذه الدالة بنسخ جزء من حرفي معلوم ، بمعلومية بداية الجزء المطلوب وعدد لبناته (أو طوله) .



وصيغة الدالة كالآتي :

### copy (string, start, length)

حيث : string الحرفي المعلوم .  
start بداية الحرفي الجزئي المطلوب نسخه .  
length طول الحرفي الجزئي (عدد اللينات) .

والبرنامج التالي يطبع اسم العائلة « عاشور » من الاسم الكامل « مصطفى عاشور » ويخصه للمتغير lastname ثم يطبعه .

```
program copy(input,output):
var name:string[30];
    lastname:string[10];
begin
    name:='MOUSTAFA ASHOOR';
    lastname:= copy(name,10,6);
    writeln(lastname)
end.
```

الاسم الكامل  
بداية الحرفي الجزئي المطلوب نسخه  
ويطوّل قدره ست لينات

البرنامج

التنفيذ

Running  
ASHOOR ← اسم العائلة

شكل (٧ - ٤٥)

(٥) إدماج كلم في جملة insert :

وهذه خاصية معروفة من خصائص برامج التحرير (editors) أو معالجة

الكلمات (word processing) حيث يمكن « حشر » كلمة أو مجموعة كلمات بداخل نص معين .

والدالة التي تؤدي هذه الوظيفة هي insert وصيغتها كالآتي :

**insert(string1 ,string2 position)**

حيث : string1 الحرفي الجزئي (الكلمة) المراد إدماجها .  
 string2 الحرفي المعلوم (الجملة) .  
 position الخانة التي يبدأ عندها الإدماج .

ونتيجة هذه الدالة لا يجوز تخصيصها لأي متغير فهي لا تنتج أية قيمة ولكنها تقوم بتغيير محتويات الحرفي string1 .

مثال (٧ - ١٨) :

```
program insert(input,output):
var names:string[30];
l:string[40];
```

البرنامج

```
begin
names:='HAZEM SALLY';
insert('and ',names,7);
writeln(names)
end.
```

الحرفي المعلوم  
 بدء اسم الخانة رقم 7  
 الكلمة المراد إدماجها

Running  
 HAZEM and SALLY

التنفيذ

النتيجة

شكل (٧ - ٤٦)

في هذا المثال كان الحرفي المعلوم هو "HAZEM SALLY" وقد تم إدماج كلمة and في الخانة السابعة فأصبح الحرفي الجديد هو :

**"HAZEM AND SALLY"**

(٦) حذف كلمة من جملة delete :

هذه الدالة تؤدي العملية العكسية للدالة insert وصيغتها كالآتي :

**delete(string ,position, length)**

حيث : string الحرفي المعلوم المراد حذف جزء منه .  
position الخانة التي يبدأ عندها الحذف .  
length طول الجزء المراد حذفه بالبنات .

مثال (٧ - ١٩) :

في هذا المثال نقدم الحالة العكسية للمثال السابق حيث نحذف الجزء and "HAZEM" من الحرفي "SALLY and HAZEM" فيصبح SALLY بعد التعديل .

```
program delete(input,output);
var names:string[30];
```

```
begin
names:='SALLY and HAZEM';
delete(names,6,10);
```

```
writeIn(names)
```

```
end.
```

البرنامج

>

Running  
SALLY

>

النتيجة

التفصيل

شكل (٧ - ٤٧)

## ■ تمارين الباب السابع :

س (٧ - ١) اكتب برنامجاً يستقبل مصفوفة من الحروف الأبجدية (بحد أقصى عشرون حرفاً) ، ويطبعها بالصورة الآتية :

```

Running
5  ← إدخال عدد الحروف
abcde ← إدخال الحروف
a
b } ← نتيجة البرنامج
c
d
e
    
```

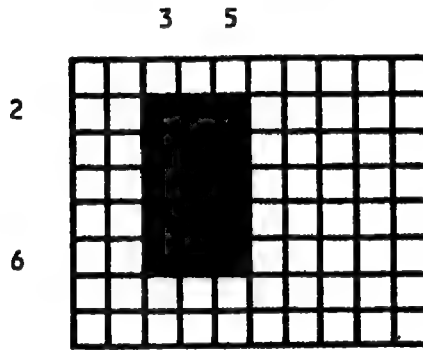
## شكل (٧ - ٤٨)

س (٧ - ٢) اكتب برنامجاً لتحليل مبيعات الأقسام لشركة من الشركات باستخدام الرسم البياني مستخدماً مقياس رسم (10) إذا كانت البيانات تستقبل في الكمبيوتر بالصورة الآتية :

المعنى	الرقم المُدخل
عملية بيع واحدة للقسم رقم 5	5
عملية بيع واحدة للقسم رقم 3	3
عملية بيع واحدة للقسم رقم 8	8
.	.
.	.
.	.
نهاية البيانات	-1

ملاحظة : تستخدم الشركة ٨ أقسام للبيع .

س (٧ - ٣) كتطوير للمثال رقم (٧ - ٨) الذى يعالج تعداد السكان ، المطلوب طباعة عدد السكان فى منطقة جزئية من المدينة التى يتم تعريفها بأربعة أرقام مثل 5 3 6 2 (أنظر شكل ٧ - ٢٠) أى المنطقة المحصورة بين الصفين 2,6 والعمودين 3,5 وقد تم تظليل المنطقة المقصورة بالرسم .



شكل (٧ - ٤٩)

س (٧ - ٤) اكتب برنامجاً يقرأ قطعة من نص تنتهى بالعلامة (\*) ويقوم البرنامج بإيجاد الحرف ذى أعلى معدل تكرار فى النص .

س (٧ - ٥) اكتب برنامجاً يقرأ عدداً صحيحاً (n) متبوعاً بكلمة تحتوى على عدد (n) من الحروف . ويقرر البرنامج إذا ما كانت الكلمة تقرأ من اليمين كما من اليسار (أى مثل كلمة ROTOR) .

س (٧ - ٦) ينقسم الأسبوع الدراسى لأحد الطلبة إلى خمسة أيام دراسية يحتوى كل منهم على ٦ فترات دراسية (محاضرات أو معامل) ويجب على الطالب

أن يحضر ٢٠ فترة خلال الأسبوع الواحد . والبرنامج المطلوب يقوم بقراءة قائمة بمواعيد المحاضرات وأماكنها حيث يتم تمثيل كل محاضرة برقمين يعبر أحدهما عن اليوم (1..5) والآخر يعبر عن الفترة (1..6) وبلى هذين الرقمين رقم ثالث يدل على قاعة المحاضرات أو المعمل .

والمطلوب من البرنامج أن يطبع الجدول الزمني للمحاضرات بالصورة الآتية :

الفترة الدراسية → period

		1	2	3	4	5	6
<div style="display: inline-block; transform: rotate(-90deg);">             أيام الأسبوع              mon              tue              wed              thu              fri           </div>	رقم القاعة	3	1	2	9		
		6	6	5	7	6	
			2	1			
		5	2	5		8	7
		3	2	1	3		

شكل (٧ - ٥٠)

س (٧ - ٧) اكتب إعلاناً عن غمط مبتكر يمثل « العام » في القرن الحالي على أن يكون الغمط صحيحاً وممثلاً بأربعة أرقام مثل ١٩٨٨ .

س (٧ - ٨) تنقسم الدراسة في أحد الكليات إلى ثلاثة ترمات (terms) الترم الأول first ، والثاني second ، والصيفي summer . اكتب إعلاناً عن الغمط term وعن متغيرين هما thisterm (الترم الحالي) و nexterm (الترم التالي) .

ثم اكتب عبارة (عبارات) لتخصيص قيمة ما للمتغير nexterm إذا أعطيت قيمة المتغير thisterm .

س (٧ - ٩) اكتب إعلاناً عن النمط "matrix" الذى يحتوى على ٣ صفوف وخمسة أعمدة من الأعداد الحقيقية .

ثم أعلن عن ثلاث مصفوفات أخرى  $c, b, a$  . وبفرض أن هناك قيماً تم تخصيصها لكل من  $b, a$  اكتب العبارات اللازمة لتوليد قيم المصفوفة  $c$  بحيث يكون كل عنصر منها يمثل مجموع العنصرين المناظرين في كل من المصفوفتين  $b, a$  .

س (٧ - ١٠) اكتب برنامجاً بلغة باسكال لحساب عدد مرات حدوث الحروف 'a' ، 'z' في ملف من اللبئات . أهمل علامات نهاية السطر واللبئات غير الأبجدية .

ثم اطبع عدد مرات تكرار كل حرف .







## الباب الثامن

### البرامج الفرعية والدوال **Procedures & Functions**



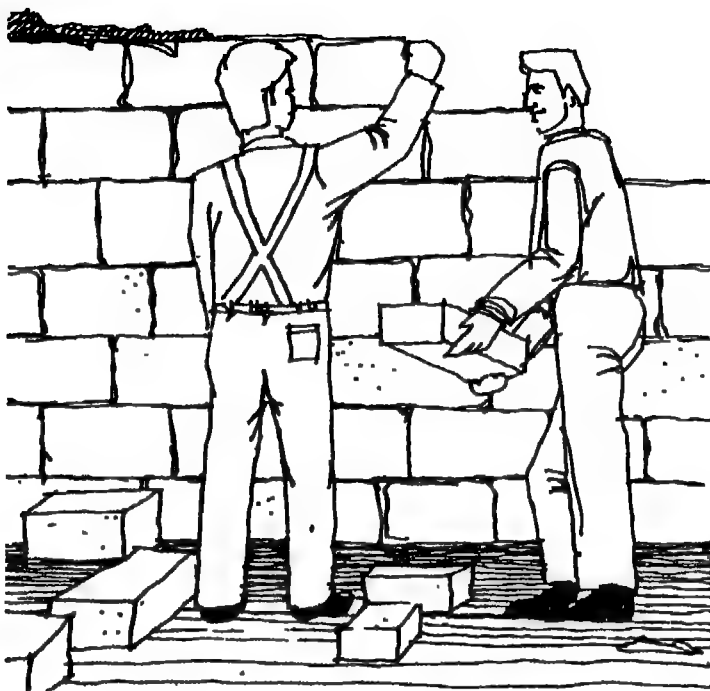
## مفتتح

إذا استعرضنا منطق أحد البرامج الكبيرة سوف نجد أنه يهدف إلى إنجاز عمل ما وهذا يحدد الصفة الأساسية المميزة للبرنامج . ولكن مهما كان البرنامج متفرداً في هذا الهدف فهو يحتوى في تفصيلاته على عمليات أخرى متكررة . وهذه العمليات المتكررة قد تكون عمليات شائعة الاستخدام مثل حساب الجذر التربيعى وجيب الزاوية أو حتى بعض العمليات الأكثر تعقيداً مثل حل معادلة رياضية أو إحصائية . وقد تكون العملية المتكررة مبتكرة ومتفردة أيضاً ينشئها المبرمج وفقاً لاحتياجاته .

وسواء كانت العملية المتكررة شائعة أو متميزة فلا معنى لتكرار برمجتها داخل البرنامج كلما احتجنا إليها . فمن المفضل في هذه الحالة أن نبرمج مرة واحدة وتستدعى عند الحاجة إليها .

وقد رأينا في الباب الثانى أن لغة باسكال تمدها بمجموعة من البرامج الجاهزة في مكتبة الدوال القياسية لكي نستخدمها في العمليات شائعة الاستخدام . أما إذا كانت العملية المتكررة مبتكرة فيمكن في هذه الحالة تقسيم البرنامج الكبير إلى وحدات منفصلة : برنامج رئيسى وبرنامج فرعية ، حيث يختص البرنامج الرئيسى بمنطق البرنامج الأصلى وتختص البرامج الفرعية بالعمليات الجانبية المتكررة .

وفي هذا الباب نتعرض لكيفية بناء البرامج باستخدام الوحدات المساعدة : البرامج الفرعية والدوال المبتكرة .



بناء البرناج من براج فرعية

شكل (٨ - ١)

## (٨ - ١) البرنامج الفرعى والبرنامج الرئيسى *procedure & program*

إذا كنا بصدد تصميم برنامج لحساب أجور الموظفين بمؤسسة ما ، فمن البديهي أن العمليات التى يتضمنها البرنامج سوف تحتوى على المعالجات الآتية :

- ١ - تجميع أجر ساعات العمل العادى لكل موظف .
- ٢ - تجميع أجر ساعات العمل الإضافية لكل موظف .
- ٣ - خصم ضريبة الدخل العام .

لا بأس عندئذ من إنشاء ثلاثة برامج صغيرة يختص كل منها بعمل من هذه الأعمال ولتكن أسماؤها بالترتيب :

**addnormalhourspay**

**addovertimehourspay**

**deductincometax**

وعند بناء البرنامج الرئيسى الذى يتولى حساب الأجور لن نحتاج أن نضمته أية خطوات لحساب الساعات الأصلية أو الإضافية إلى آخره بل كل ما نحتاجه أن نكتب اسم أحد هذه البرامج الفرعية بداخل البرنامج الرئيسى ، فيتولى البرنامج الرئيسى استدعاءها وتنفيذها . عندئذ يصير البرنامج الفرعى كأنه عبارة من عبارات باسكال وعادة يكتب البرنامج الفرعى (procedure) بداخل البرنامج الرئيسى (program) بعد إعلان المتغيرات مباشرة ، أى قبل بداية البرنامج الرئيسى (Begin) .

والشكل (٨ - ٢) يوضح علاقة البرنامج الفرعى B بالبرنامج الرئيسى A كشكل عام .

```

program A(input,output);
VAR.....

.....          البرنامج الرئيسى A

procedure B;
VAR.....
begin
.....          البرنامج الفرعى B
end;

begin
...
B      ←          استخدام البرنامج الفرعى
                  كخطوة من خطوات المعالجة
...
end

```

شكل (٨ - ٢)

مثال (٨ - ١) :

ينتج مصنع للأدوات الميكانيكية « محاور » ذات أقطار مختلفة من ماكيتى إنتاج . وفى خط الإنتاج تؤخذ عينة مكوّنة من عشرة محاور من الماكينة الأولى ويتم حساب متوسط أقطارها ثم تؤخذ عينة من الماكينة الثانية ويحسب متوسط أقطارها . وفى النهاية يتم حساب المتوسط العام للعشرين .

والبرنامج التالى يقوم بهذه العملية وهو يتكون من برنامج رئيسى **average20** وبرنامج فرعى **average10** .

وكما هو واضح من أسماء البرامج فالبرنامج الفرعى يختص بحساب المتوسط لعشرة قيم ، أما الرئيسى فهو يحسب المتوسط النهائى لمجموعتين من العينات مكونة من ٢٠ قطعة . شكل (٨ - ٣) :

program average20(input,output);

var overallsum : real; البرنامج الرئيسى

```

procedure average10;
var count : integer; next, sum10 : real;
begin
    sum10 := 0;
    for count := 1 to 10 do
        begin
            read(next);
            sum10 := sum10 + next;
        end;
        writeln('average of 10 readings : ', sum10/10);
        overallsum := overallsum + sum10;
    end;

```

البرنامج الفرعى

```

begin
    overallsum := 0;
    average10;
    average10;
    writeln('overall average : ', overallsum/20);
end.

```

شكل (٨ - ٣)

وكما نرى فإن البرنامج الرئيسى يبدأ بالفعل بعد البرنامج الفرعى أى أن تنفيذ البرنامج يبدأ بالعباراة :

**overallsum:= 0**

وهي العبارة الأولى بالبرنامج الرئيسى . يلى ذلك العبارة الثانية :

**average 10**

وهذه العبارة هي نفسها اسم البرنامج الفرعى وينتج عنها استدعاء وتنفيذ البرنامج الفرعى لذلك فهي تسمى عبارة نداء (procedure call) . والعبارة الثالثة هي أيضاً عبارة نداء لنفس البرنامج الفرعى . أما العبارة الأخيرة فهي تحسب المتوسط الكلى للعينات وتطبعه بالتعبير :

**overallsum/20**

ولنلق نظرة أكثر قرباً على متغيرات البرنامج :

### ● المتغيرات العامة Global variables

في فقرة الإعلانات بالبرنامج الرئيسى نلتقى بالمتغير overallsum وهذا المتغير يعتبر متغيراً عاماً (global variable) بمعنى أنه سيجوز استخدامه في أى جزء من البرنامج .

### ● المتغيرات المحلية Local variables :

أما في بداية البرنامج الفرعى فنرى إعلاناً عن المتغيرات count, next, sum10 وهذه النوعية من المتغيرات لا يجوز أن تستخدم إلا في البرنامج الفرعى . ولذلك يطلق عليها المتغيرات المحلية (local variables) .

والسبب في ذلك أن هذه المتغيرات تحتل خانات الذاكرة فقط أثناء استدعاء البرنامج الفرعى الذى يستخدمها فإذا انتهى تنفيذ البرنامج الفرعى يتم مسح هذه الخانات من الذاكرة توطئة لا استخدامها مع بيانات البرنامج الرئيسى .

ولذلك فإن على المبرمج أن يقرر قبل كتابة البرنامج : أى المتغيرات سوف يحتاج إليها أثناء البرنامج الرئيسى حتى يعلنها كمتغيرات عامة أما المتغيرات التى



تخدم البرنامج الفرعى فيكتفى بإعلانها بداخل البرنامج الفرعى فقط كمتغيرات محلية . وهذا الأسلوب يجنب استهلاك الذاكرة بلا داع .

## ما فائدة البرنامج الفرعى ؟ .

لعلنا نرى فى هذا المثال أننا كتبنا البرنامج الفرعى مرة واحدة واستدعيناه مرتين ، وفى بعض التطبيقات قد نحتاج استدعاء البرنامج الفرعى عشرات المرات . لذلك فقد أغنانا وجود البرنامج الفرعى عن تكرار كتابته عدة مرات فى نفس البرنامج .

هذا فضلاً عن أنه يجعل البرنامج سهل القراءة على الإنسان فالبرنامج بهذه الصورة يتكون من أربع عبارات فقط سهلة القراءة سهلة الفهم .

## (٨ - ٢) البارامترات : Parameters

فى المثال السابق رأينا أن البرنامج الفرعى يقوم بتنفيذ عملية محدّدة فى كل مرة ، يستقبل قيم نفس المتغيرات ، يعالجها ، ويطبع النتائج . ولكن هناك وسيلة تجعل البرنامج الفرعى أكثر مرونة بحيث يمكنه تنفيذ عمليات مختلفة أو التعامل مع متغيرات مختلفة . يتم ذلك بإمداد البرنامج الفرعى ببارامتر parameter . ولنضرب مثلاً .

## مثال (٨ - ٢) :

لنفرض أننا نريد كتابة برنامج لطبع فاتورة الكهرباء حيث يستقبل عددين يمثلان القراءة الحالية والقراءة السابقة (يحتوى كل عدد على أربعة أرقام) ثم يطبع القراءتين والمبلغ المطلوب . هذا المنطق قد يكون كالآتى :

١ — اقرأ القراءة السابقة (previous) والقراءة الحالية (present) .

٢ — اطبع العناوين « القراءة السابقة » ، « القراءة الحالية » ، « المبلغ المطلوب » .

- ٣ — اطبع القراءة السابقة .
- ٤ — اطبع القراءة الحالية .
- ٥ — اطبع المبلغ المطلوب .

ولأن الأعداد المكتوبة في فاتورة الكهرباء عادة ما تكون رباعية الأرقام (أى أن العدد 45 على سبيل المثال يكتب على الصورة 0045) فلعله من المناسب أن نطلق الاسم **"write4digit"** على البرنامج الفرعى الذى يتولى كتابة قراءة العداد .

فلنكتب الآن البرنامج الفرعى الذى يمكن استخدامه مرةً لطباعة القراءة الحالية ومرةً أخرى لطباعة القراءة السابقة .. وشكل (٨ — ٤) يوضح صورة ممكنة لهذا البرنامج .

```

procedure write4digits(meterreading : integer);
begin
    write(' ');
    if meterreading < 10 then write('000')
    else if meterreading < 100 then write('00')
    else if meterreading < 1000 then write('0');
    write(meterreading) ;
end;
    
```

البارامتر

### شكل (٨ — ٤)

إن اسم البرنامج الفرعى — كما اتفقنا عليه — هو : **"write4digit"** ولكن كلمات جديدة ظهرت ما بين القوسين عقب الاسم مباشرة وهى :

**(meterreading:integer);**

أما الاسم **meterreading** فيسمى ( البارامتر ) (وهو من النوع الصحيح فى هذه الحالة وفقاً للإعلان integer) . وعندما يستدعى البرنامج

الفرعى فإن هذا البارامتر يأخذ قيمة جديدة وهى القيمة المطلوب استخدامها فى البرنامج الفرعى بالفعل . فإذا استدعينا هذا البرنامج الفرعى بعبارة مثل :

### **write4digits(previous)**

فإن البارامتر meterreading يُستبدل بالمتغير previous وعليه فإن البرنامج الفرعى يطبع قيمة القراءة السابقة .

وأما إذا استدعيناها بالعبارة :

### **write4digits(present)**

فإن المتغير present يستبدل البارامتر meterreading ويقوم البرنامج الفرعى بطبع قيمة القراءة الحالية .

وعادة ما يسمى المتغير present (أو المتغير previous) الذى يستخدم فى الاستدعاء بالبارامتر الحقيقى أما البارامتر بالبرنامج الفرعى فقد يسمى بالبارامتر الهيكلى .

ويجوز أن يحل محل البارامتر الهيكلى أى عدد صحيح (ثابت صحيح أو تعبير صحيح) إذ أنه ليس بالضرورة أن يكون متغيراً ، كالمثال الآتى :

### **write4digits(256 + 17)**

فى هذه الحالة يكون الخرج هو العدد 0273 .

وشكل (٨ - ٥) يوضح الخطوات الأساسية من البرنامج الرئيسى التى تستدعى بها البرنامج الفرعى لطباعة فاتورة الكهرباء .

```
read(previous, present); // للقراءة السابقة
writeln('prev reading    pres reading    total due');
write4digits(previous); write(' ');
write4digits(present); // للقراءة الحالية
writeln('f':8, (present - previous)*unitprice :5:2)
```

شكل (٨ - ٥)

أما شكل (٨ - ٦) فيوضح الشكل النهائي للبرنامج الرئيسي متضمناً  
البرنامج الفرعي write4digit .

```

program electricitybill(input,output);

const unitprice = 0.034;

var previous, present : integer;

procedure write4digits(meterreading : integer);
begin
    write(' ');
    if meterreading < 10 then write('000')
    else if meterreading < 100 then write('00')
    else if meterreading < 1000 then write('0');
    write(meterreading)
end;

begin
    read(previous, present);
    writeln('prev reading    pres reading    total due');
    write4digits(previous); write(' ');
    write4digits(present);
    writeln('£':8, (present - previous)*unitprice :5:2)
end.

```

البرنامج الرئيسي

البارامترات الهيكلية

البرنامج الفرعي

البارامترات الحقيقية

شكل (٨ - ٦)

(٨ - ٣) استخدام أكثر من بارامتر :

لا مانع من استخدام أكثر من بارامتر هيكلية في نفس البرنامج الفرعي ولا  
بأس من استخدام بارامترات من أنماط مختلفة .

ولنلق نظرة على البرنامج الفرعي شكل (٨ - ٧) .

```

procedure writechar(ch : char; nooftimes : integer);
var count : integer;
begin
  for count:= 1 to nooftimes do write(ch);
  writeln
end;

```

البارامترات  
عدد مرات التكرار  
اللبنة المراد طبعتها

### شكل (٨ - ٧)

هذا البرنامج الفرعى يستقبل عدداً صحيحاً ولبنة ، ثم يقوم بطباعة اللبنة عدداً من المرات يساوى العدد الصحيح الذى تم استقباله . فإذا أردنا استدعاء هذا البرنامج الفرعى فلنكتب عبارة مثل :

**writechar('\*', 5)**

عند تنفيذ هذا الاستدعاء سوف يقوم البرنامج الفرعى بطباعة خمسة نجوم متجاورة حيث أن اللبنة (\*) سوف تحل محل البارامتر ch والعدد 5 سوف يحل محل البارامتر nooftimes . والنقطة الجديرة بالملاحظة هنا هي أن البارامترات الهيكلية ظهرت مفصولة بفاصلة منقوطة في البرنامج الفرعى لأنها تنتمى إلى أنماط مختلفة .

أما في عبارة الاستدعاء فقد ظهرت البارامترات الحقيقية مفصولة بفاصلة عادية .

وكما ذكرنا من قبل يمكن أن تكون البارامترات الحقيقية المستخدمة في استدعاء البرنامج الفرعى — تعبيرات أو متغيرات من نفس النوع . فإذا كان لدينا متغير لبنة اسمه **symbol** (بمعنى رمز) ومتغير صحيح اسمه **noofsymbols** (بمعنى عدد الرموز) ، فيمكننا كتابة العبارات التالية على سبيل المثال (٨ - ٨) .

```
symbol := 'a'; noofsymbols := 5;
writechar(symbol, noofsymbols);
writechar(succ(succ(symbol)), 2*noofsymbols + 1)
```

### شكل (٨ - ٨)

سيُنتج عن هذه العبارات طباعة الحرف a خمس مرات نتيجة لتنفيذ العبارة الثانية ، والحرف c إحدى عشر مرة نتيجة لتنفيذ العبارة الثالثة (تذكر معنى الدالة succ) .

أى أن الخرج سيكون :

aaaaa

ccccccccccc

ماذا لو كانت البارامترات الهيكلية من نفس النوع (النمط) ؟ .

إنها في هذه الحال يمكن أن تظهر مفصولة بفاصلة عادية . والبرنامج الفرعى التالى (شكل ٨ - ٩) يوضح مثلاً لاستخدام مجموعات من البارامترات متشابهة وغير متشابهة .

```
procedure demo(i,j :integer; a,b,c,d :char; x,y,z :real);
begin
  .
  .
  .
end;
```

صحيح      لبنية      حقيقي

لاحظ الفاصلة والفاصلة المنقوطة

### شكل (٨ - ٩)

أما عند الاستدعاء فتفصل جميع البارامترات الحقيقية بفاصلة عادية كالمثال  
الآتي :

demo(256, 73, '\*', 'p', 'q', 'r', 26.45, 72.6, 18.253)

### مثال (٨ — ٣) البرنامج الابن والبرنامج الحفيد ا

في هذا المثال يقرأ البرنامج عددين صحيحين مرجبين ويطبعهما في ترتيب  
تنازلي كما يضيف إلى العدد علامات الفاصلة لتمييز خانة الآلاف والملايين فالعدد  
**1235548** يكتب على الصورة **1,235,548** .

```
program compare(input,output);
```

```
var first, second : integer;
```

```
procedure commawrite(n : integer);
var millions, thousands, units : integer;
```

```
procedure zerowrite(m : integer);
begin
  if m < 100 then write('0');
  if m < 10 then write('0');
  write(m)
end;
```

البرنامج الفرعي المحفّيد

```
begin
  millions := n div 1000000;
  thousands := (n mod 1000000) div 1000;
  units := n mod 1000;
  if millions > 0 then
    begin
      write(millions, ',');
      zerowrite(thousands); write(',');
      zerowrite(units)
    end
  else
    if thousands > 0 then
      begin
        write(thousands, ',');
        zerowrite(units)
      end
    else write(units);
  end
  writeln
end;
```

البرنامج الفرعي الـ ١٠

```
begin
```

```
  read(first, second);
  if first > second then
    begin
      commawrite(first); commawrite(second)
    end
  else
    begin
      commawrite(second); commawrite(first)
    end
  end
end.
```

البرنامج الرئيسي

شكل (٨ - ١٠)



ومن المفضل عند قراءة برنامج كبير أن نبدأ بقراءة البرنامج الرئيسى (وكذلك عند كتابة البرنامج). فالبرنامج الرئيسى يبدأ بقراءة العددين **first, second** ثم يقارن بينهما باستخدام العبارة الشرطية لترتيبها تنازلياً. ثم يتولى البرنامج الفرعى commawrite طباعة الفواصل. عند إنشاء مثل هذا البرنامج نبدأ بهذه الخطوات المنطقية ثم نفكر بعد ذلك فى خطوات البرنامج الفرعى الذى سيطبع الفواصل.

وعند إنشاء البرنامج الفرعى commawrite سوف تتضح الحاجة إلى برنامج فرعى آخر بداخل البرنامج الفرعى الأول وهو البرنامج zerowrite وذلك لوجود أرقام صغيرة مثل 32,152 لا يتطلب طباعتها محتوية أصفاً على يسارها.

والمفهوم الذى نخرج به من هذا المثال أن البرامج الفرعية قد تتداخل وتتعدد بحيث يكون هناك برنامج رئيسى وبرنامج ابن وبرنامج حفيد وهكذا.. وينطبق على البرنامج الابن والبرنامج الحفيد ما ينطبق على البرنامج الرئيسى والبرنامج الفرعى من حيث أن متغيرات البرنامج الابن تعتبر متغيرات عامة بالنسبة للبرنامج الحفيد ويمكن استخدامها بداخله.

وهذا لا يمنع أن تكون هناك عدة برامج فرعية غير مرتبطة ببعضها البعض وكلها تخدم برنامجاً رئيسياً واحداً كالمثال التالى:

#### مثال (٨ — ٤) ترجمة لغات الكمبيوتر:

نعلم أن لغات الكمبيوتر عالية المستوى — مثل باسكال — تحتاج إلى مترجم compiler لتحويل التعليمات المكتوبة بهذه اللغة إلى لغة الماكينة التى تتسم بالتبسيط الشديد والوصف التفصيلي لكل خطوة من الخطوات المطلوبة من الكمبيوتر. فعلى سبيل المثال إذا اعتبرنا العملية الحسابية الآتية:

$$a + x * z$$

فلكى نصف تنفيذ هذه العملية للكمبيوتر باستخدام لغة الماكينة فإننا نصفها بطريقة مماثلة للأسلوب الذى نشرح به خطوات استخدام الماكينة الحاسبة كالآتى :

- ١ — أدخل قيمة المتغير x .
- ٢ — اضغط على الزر (\*) (زر الضرب) .
- ٣ — أدخل قيمة المتغير z .
- ٤ — اضغط على زر الجمع (+) .
- ٥ — أدخل قيمة المتغير y .
- ٦ — اطبع قيمة الناتج على الشاشة .

لاحظ أولوية عملية الضرب !

وللتسهيل سوف نفترض هذا العدد المحدود من المتغيرات والمؤثرات كما نفترض عدم استخدام الأقواس . ويتكون المتغير من حرف واحد بلا مسافات خالية بينه وبين أى لبنة أخرى .

وعلى البرنامج أن يحدد أى المؤثرات يتعامل معها قبل الآخر (الأولوية) ، فالضرب والقسمة يأتیان قبل الجمع والطرح . وإذا تساوت أولوية مؤثرين فإن الأولوية تعطى لمن يأتى أولاً .

ويمكن التعبير عن هذا المنطق كالآتى :

إذا كان المؤثر الثانى أحد عناصر الفقة ['\*', '/', '+', '-']  
والمؤثر الأول أحد عناصر الفقة ['+', '-', '\*', '/'] .  
إذن فنفذ العملية المصاحبة للمؤثر الثانى قبل العملية المصاحبة للمؤثر الأول .  
وإلا فنفذ العملية المصاحبة للمؤثر الأول قبل العملية المصاحبة للمؤثر الثانى .

وهذا المنطق يمكن وصفه بشريحة البرنامج التالية :

```

if (op2 in ['*', '/']) and (op1 in ['+', '-']) then
begin
    valueof(var2);
    apply(op2);
    valueof(var3);
    apply(op1);
    valueof(var1)
end
else
begin
    valueof(var1);
    apply(op1);
    valueof(var2);
    apply(op2);
    valueof(var3)
end

```

الشروط

إجراء العملية الثانية

إجراء العملية الأولى

### شكل (٨ - ١١)

في هذه الشريحة من البرنامج وصفنا المؤثر الأول بالمتغير **op1** والمؤثر الثاني بالمتغير **op2** . كما نلاحظ استخدام الفاصلة مع الفئة التي تحتوي على عناصر غير متصلة مثل ['+', '-', '\*'].  
 فإذا أتينا إلى تفصيلات تنفيذ العملية المصاحبة للمؤثر الثاني (العملية الثانية) نجد أن أول خطوة هي :

**valueof(var2)**

وبالطبع نتوقع أن يكون **valueof** هو اسم برنامج فرعى يقوم بإدخال قيمة المتغير **var2** . كما نلاحظ أن اسم المتغير قد استخدم كإبرامتر تمهيداً لاستبداله بإبرامتر حقيقي .

أما العبارة **apply** فهي تناظر برنامجاً فرعياً آخر يقوم بإصدار التعليمات

للضغط على زر معيّن من أزرار المؤثرات (/، -، +، \*) والمؤثر المستخدم مع هذا البرنامج الفرعي هو البارامتر op2 . وهكذا ...  
ويوضح شكل (٨ — ١٢) البرنامج الكامل الذي يصف هذه العملية .

```
program evaluate(input, output);
```

```
var var1, op1, var2, op2, var3 : char;
```

```
procedure valueof(variable : char);
```

```
begin
```

```
    writeln('key in the value for ', variable)
```

```
end;
```

أدخل قيمة المتغير

```
procedure apply(op : char);
```

```
begin
```

```
    write('press the ');
```

```
    case op of
```

```
        '+': write('addition');
```

```
        '-': write('subtraction');
```

```
        '*': write('multiplication');
```

```
        '/': write('division')
```

```
    end;
```

```
    writeln(' key')
```

```
end;
```

اصنع على زر العملية المطلوبة

```
begin
```

```
    read(var1, op1, var2, op2, var3);
```

```
    if (op2 in ['*', '/']) and (op1 in ['+', '-']) the
```

```
    begin
```

```
        valueof(var2);
```

```
        apply(op2);
```

```
        valueof(var3);
```

```
        apply(op1);
```

```
        valueof(var1)
```

```
    end
```

```
    else
```

```
    begin
```

```
        valueof(var1);
```

```
        apply(op1);
```

```
        valueof(var2);
```

```
        apply(op2);
```

```
        valueof(var3)
```

```
    end;
```

```
    writeln('display the answer.')
```

```
end.
```

شكل (٨ - ١٢)

ترجمة لغات الكمبيوتر

### (٨ - ٤) البارامتر المتغير variable parameter :

في الفقرة السابقة كان البارامتر دائماً يصف قيمة معينة تتم معالجتها عندما يستدعى البرنامج الفرعي كالمثال الآتي :

```
procedure process(x : integer);
begin
    writeln(x);
    writeln(x*x)
end;
```

### شكل (٨ - ١٣)

وعندما يستدعى هذا البرنامج الفرعي فإنه استبدال البارامتر x بأي تعبير صحيح مثل المتغير (a) أو قيمته العددية أو أي عنصر من عناصر مصفوفة عددية مثل mark[2,3] (كما في شكل ٨ - ١٤) :

```
process(5742);
process(a);
process(a*5 + 32);
process(mark[2,3]);
process(mark[2,3]*3. + 564)
```

### شكل (٨ - ١٤)

وعندما يستدعى البرنامج الفرعي بإحدى هذه العبارات فإن البارامتر الحقيقي يُرسل البيان المطلوب إلى البرنامج الفرعي ، ويتم المعالجة .  
هذه النوعية من البارامترات يطلق عليها البارامترات البسيطة (simple)

## . parameters)

ولكنه في بعض الأحيان قد يتطلب الأمر إرسال البيانات من البرنامج الفرعى إلى البرنامج الرئيسى ، وهذا يمكن تحقيقه بتزويد البرنامج الفرعى ببارامتر متغير (variable parameter) والذي يخبر البرنامج الفرعى : « إلى أين يرسل المعلومة المطلوبة بالتحديد » . ولنضرب مثلاً بالبرنامج الذى عرضناه في التمرين (٤ — ١٠) عن جمع الأصوات في الانتخابات (أنظر إجابة التمارين في نهاية الكتاب) ، ولنستمر في تطوير البرنامج حتى يطبع رسالة باسم الحزب الفائز .

والمنطق التالى يعبر عن البرنامج المطلوب :

١ — اجمع الأصوات لحزب واحد وخزن الناتج في المتغير  
**party 1 overall**

٢ — اجمع الأصوات للحزب الثانى وخزن الناتج في المتغير  
**party 2 overall**

٣ — إذا تساوى عدد الأصوات لكل من الحزبين اطبع النتيجة التالية  
 «a draw» بمعنى تعادل !.

٤ — وإلا فإذا كان عدد أصوات الحزب الأول أكبر فاطبع النتيجة «a win for party 1.» أى فوز الحزب الأول .

٥ — وإلا فاطبع النتيجة «a win for party 2.» أى فوز الحزب الثانى .

ولو تأملنا الخطوتين (١) ، (٢) لوجدنا أن العملية المطلوبة بكل منهما واحدة وهى جمع الأصوات وتخزينها في متغير والفارق الوحيد هو اسم المتغير الذى توضع فيه النتيجة .

فلنتشبه إذن برنامجاً فرعياً، وليكن اسمه “**addupvotesfor**” بمعنى « اجمع الأصوات للحزب ... » ولنتوقع أن نستخدمه بالطريقة الآتية :

**addupvotesfor(party 1 overall);**

**addupvotesfor(party 2 overall)**

في الحالة الأولى يُستخدم لتجميع أصوات الحزب الأول .  
 وفي الحالة الثانية يُستخدم لتجميع أصوات الحزب الثاني .  
 أى أننا نطلب من البرنامج الفرعى أن يضع قيمة معينة في المتغير المستخدم  
 كبارامتر . وهذا يجب الإعلان عنه في تعريف البرنامج الفرعى كما هو موضح  
 بالمثال الآتى :

مثال (٨ - ٥) انتخابات ..

```

program election3(input,output);
var party1overall, party2overall : integer;

procedure addupvotesfor(var total : integer);
var next : integer;
begin
    total := 0;
    read(next);
    repeat
        total := total + next;
        read(next)
    until next < 0
end;

begin
    addupvotesfor(party1overall);
    addupvotesfor(party2overall);

    if party1overall = party2overall then
        writeln('a draw.')
    else if party1overall > party2overall then
        writeln('a win for party 1.')
    else writeln('a win for party 2.')
end.
    
```

البارامتر المتغير

البرنامج الفرعى لتجميع أصوات الأحزاب

شكل (٨ - ١٥)



وكما نلاحظ في مستهل البرنامج الفرعى فإن كلمة **VAR** قد ظهرت في تعريف البارامتر **total** .

ولذلك فعندما يُستدعى البرنامج الفرعى بالعبارة :

**addupvotesfor(party1overall)**

فإن المتغير **total** سوف يشير إلى نفس خانة الذاكرة التى يشير إليها المتغير **party1overall** .

ولذلك فإن تنفيذ العبارة الواردة في البرنامج الفرعى :

**total := total + next**

يكافئ تماماً كما لو أن العبارة التالية هى التى نفذت :

**party1overall := party1overall + next**

أى أن الأصوات يتم تجميعها في المتغير **party1overall** .

كذلك الحال بالنسبة لتجميع الأصوات في المتغير الثانى **party2overall** .

ملاحظات :

- من الجدير بالذكر أنه بخلاف البارامترات البسيطة لا يجوز استدعاء البرنامج الفرعى باستخدام بارامتر ثابت مثل :

**addupvotesfor(3);**

أو باستخدام تعبير مثل :

**addupvotesfor(party1overall + party2overall)**

فالبرنامج الفرعى ينتظر منا اسم متغير (أو اسم متغير دليل) حتى يضع فيه القيمة الناتجة من المعالجة .

فعلى سبيل المثال كان من الجائز إعلان مصفوفة مثل :

**var party : array [1..3] of integer;**

وفى هذه الحال يمكن استدعاء البرنامج الفرعى بعبارات كالعبارات الآتية :

**addupvotesfor(party[1]);**

**addupvotesfor(party[2]);**

**addupvotesfor(party[3])**

أو باستخدام حلقة تكرارية مثل :

**for next := 1 to 3 do**

**addupvotesfor(party[next])**

● ومن الملاحظات التى يجب وضعها فى الاعتبار هى ضرورة فصل البارامترات المتغيرة عن بقية البارامترات بفاصلة منقوطة . والمثال الآتى يعرض برنامجاً فرعياً يستقبل قيمتين **value 1** ، **value 2** ويضع مجموعهما فى متغير ثالث **sum** وحاص ضربهما فى متغير رابع **product** شكل (٨ - ١٦) :

```
procedure addandmultiply(value1, value2 : real;
                        var sum, product : real);
begin
    sum := value1 + value2;
    product := value1 * value2
end;
```

لاحظ الفاصلة المنقوطة

شكل (٨ - ١٦)

ورغم أن البارامترات جميعاً حقيقية لكننا نلاحظ تقسيمها إلى بارامترات بسيطة ومتغيرة وفصل المجموعتين بفاصلة منقوطة .

وقد يستدعى مثل هذا البرنامج الفرعى بالطريقة الآتية بعد (شكل ٨ - ١٧) :

```
var x, y, sum1, prod1 : real;
    a, b : array [1..10] of real;
.
.
.
addandmultiply(3.4, 7.2, x, y);
addandmultiply(x + 3.7, y + 4.3, sum1, prod1);
addandmultiply(23.72, 63.15, a[1], b[1]);
.
.
```

شكل (٨ - ١٧)

#### (٨ - ٥) البارامترات الدليلية *array parameters* :

عرفنا من قبل المتغيرات الدليلية وهى المتغيرات التى تمثل عناصر المصفوفة . وفى هذه الفقرة نعرض استخدام المتغيرات الدليلية كبارامترات للبرامج الفرعية . وقبل أن نعرض تفاصيل الطريقة المستخدمة دعنا نرى أولاً الحاجة التى تدفعنا لاستخدام مثل هذه النوعية من البارامترات . ولنفرض أن لدينا درجات مجموعة من الطلبة تقدموا للامتحان فى ست مواد مختلفة ، والمطلوب كتابة برنامج يقوم بقراءة وحساب درجات المواد المختلفة والمجموع لكل طالب وطباعة هذه النتائج بحسب الترتيب : الأعلى فالأقل . ولنبدأ بطالين هما على وعمر : وفى هذه الحالة يمكن توقع أن يعمل البرنامج كما فى شكل (٨ - ١٨) حيث يستقبل درجات كل من عمر وعلى ثم يطبع النتائج مرتبة .

Running

8  
7  
6  
7  
8  
5  
4  
7  
8  
8  
7  
9

درجات الطالب "عمر"

درجات الطالب "علي"

تنفيذ البرنامج المتوقع :-

المجموع الكلي .. بالترتيب

ALY: 4 7 8 8 7 9  
OMAR: 8 7 6 7 8 5

total: 43  
total: 41

>

شكل (٨ - ١٨)

ولنبدأ بتصميم البرنامج الرئيسي .. وليكن هو البرنامج الموضح شكل

(٨ - ١٩) .

```

var omarsmarks, alysmarks : array [1..6] of integer;
    omarstotal, alystotal : integer;
.
.
.
begin
    readandadd(omarstotal, omarsmarks);
    readandadd(alystotal, alysmarks);
    if omarstotal > alystotal then
        begin
            write('OMAR: '); writeout(omarstotal, omarsmarks);
            write('ALY: '); writeout(alystotal, alysmarks);
        end
    else
        begin
            write('ALY: '); writeout(alystotal, alysmarks);
            write('OMAR: '); writeout(omarstotal, omarsmarks);
        end
    end
end.

```

البرنامج الفرعي لقراءة وتجميع الدرجات

مجموع درجات عمر

مجموع درجات عمر

البرنامج الرئيسي

## شكل (٨ - ١٩)

وفي هذا البرنامج فإن وظيفة البرنامج الفرعي readandadd هي قراءة درجات المواد الستة وتخزينها في مصفوفة الدرجات الخاصة بكل طالب ثم تجميع هذه الدرجات في متغير مجموع الدرجات لكل طالب .

omarsmarks	فمصفوفة درجات الطالب عمر هي :
omarstotal	أما متغير مجموع الدرجات له فهو
alysmarks	ومصفوفة الدرجات للطالب على هي
alystotal	ومتغير مجموع الدرجات هو

ولذلك فإن البرنامج الفرعي سوف يتم تعريفه بواسطة هذين البارامترين :  
مصفوفة من ستة عناصر ، ومتغير صحيح .

ولكن في مثل هذه الحالة لا يجوز استخدام التوصيف المعقد `array[1..6]` لتوصيف بارامتر البرنامج الفرعى . بل يجب استخدام الأنماط المبكرة بإضافة سطر جديد مثل :

**type marklist = array [1..6] of integer :**

في هذا الإعلان تم التعريف بالمصفوفة `marklist` التى تصلح لاستخدامها خلال البرنامج كمصفوفة عامة يمكن أن تحتوى درجات أى طالب .  
وبذلك فإن البرنامج الفرعى لقراءة وتجميع الدرجات يصبح كالآتى :

```

procedure readandadd (var markfor : marklist ;
                      var total : integer);
var nextpaper : integer ;
begin
    total := 0;
    for nextpaper := 1 to 6 do
        begin
            readln (markfor[nextpaper]);
            total := total + markfor[nextpaper];
        end
    end;

```

*مصفوفة عامة*  
*البرنامج الفرعى لقراءة وتجميع الدرجات*

شكل (٨ - ٢٠)

كما أن المتغيرات المعلنة في البرنامج الرئيسى يمكن تبسيطها إلى الصورة التالية :

**var omarsmarks, alysmarks : marklist ;**  
**omarstotal, alystotal : integer ;**

أى أن بارامتر المصفوفة أو البارامتر الدليلى يجب أن يكون دائماً في صورة متغير **VAR** (لأ في حالات قليلة تخرج عن حدود الكتاب) .

وهذا هو البرنامج الكامل باستخدام البارامترات الدليلية :

```
program exam ' input, output;
```

البرنامج الرئيسي

```
type marklist = array [1..6] of integer ;
```

```
var omarsmarks, alysmarks : marklist ;
    omarstotal, alystotal : integer ;
```

```
procedure readandadd (var markfor : marklist ;
                      var total : integer) ;
var nextpaper : integer ;
begin
    total := 0;
    for nextpaper := 1 to 6 do
    begin
        readln (markfor[nextpaper]);
        total := total + markfor[nextpaper];
    end
end;
```

البرنامج الفرعي لتجميع الدرجات

```
procedure writeout (var markfor : marklist;
                   total : integer);
var nextpaper : integer;
begin
    for nextpaper := 1 to 6 do
        write(markfor[nextpaper] : 3);
        writeln(' total: ', total)
    end;
```

البرنامج الفرعي للطباعة

```
begin
    readandadd(omarsmarks, omarstotal);
    readandadd(alysmarks, alystotal);
    if omarstotal > alystotal then
    begin
        write('OMAR: '); writeout(omarsmarks, omarstotal);
        write('ALY: '); writeout(alysmarks, alystotal);
    end
    else
    begin
        write('ALY: '); writeout(alysmarks, alystotal);
        write('OMAR: '); writeout(omarsmarks, omarstotal);
    end
end.
```

شكل (٨ - ٢١)

## (٨ - ٦) الدوال *functions* :

عرفنا من قبل الدوال القياسية الموجودة في لغة باسكال مثل `sqrt` ، `sqr` وفي هذه الفقرة سوف نعرض الدوال المبتكرة أى التى يقوم بإنشائها المبرمج وفقاً لاحتياجاته . والدالة تؤدي عملاً مشابهاً للبرنامج الفرعى ولكن لكل منهما استخداماً مناسباً . فلنر الفارق أولاً ما بين الدالة والبرنامج الفرعى .

تشابه الدوال مع البرامج الفرعية في أن كليهما يمثل جزءاً مستقلاً من وحدات بناء البرنامج (`module`) ويتم استدعاؤه باستخدام اسم ما . ولكن طريقة الاستدعاء تختلف لكل منهما . أما الفارق الأساسى بين النوعين فهو أن الدالة تستخدم لتوليد قيمة وحيدة تحل محل التعبير الذى استدعيت به الدالة .

ولنضرب مثلاً باستخدام الدالة القياسية `sqrt` في التعبير الآتى :

$$y := x + \text{sqrt}(2)$$

عند تنفيذ هذه العبارة فإن الدالة `sqrt` تقوم بتوليد قيمة عددية معينة تحل محل التعبير `sqrt(2)` في العبارة .

فإذا كانت الدالة مبتكرة ، فسوف تؤدي عملها بنفس الطريقة فيما عدا أن المبرمج سوف يضيف إلى برنامجه برنامجاً صغيراً لتعريف الدالة المبتكرة ووصف العمل الذى تقوم به .

ولنضرب مثلاً ....

مثال (٨ - ٦) :

البرنامج الآتى يستقبل ثلاثة أزواج من الأرقام ويقوم بجمع الأرقام الكبرى في كل زوج من الأزواج . شكل (٨ - ٢٢) .



```
program add(input,output);
```

```
var a,b,p,q,x,y : real;
```

```
function max(first,second : real) : real;
```

```
begin
```

```
  if first > second then max := first
```

```
  else max := second
```

```
end
```

برنامج الدالة

```
begin
```

```
  read(a,b,p,q,x,y);
```

```
  writeln(max(a,b) + max(p,q) + max(x,y))
```

```
end.
```

بداية البرنامج الرئيسي

## شكل (٨ - ٢٢)

وفي هذا البرنامج نرى برنامج الدالة قد جاء مباشرة بعد فقرة الإعلانات — تماماً كما البرنامج الفرعى — ووظيفة برنامج الدالة (أو الدالة) هنا هي مقارنة عددين first ، second وإيجاد أكبرهما ووضعهما في المتغير max الذى يمثل اسم الدالة .

ونلاحظ في البرنامج الرئيسى أن استدعاء الدالة يتم باستخدام الاسم max الذى يعقبه بارامتران مثل :

**max(x,y)**

وعندما يتم تقييم هذا التعبير فإنه يستبدل بالكامل بالقيمة التى أنتجها برنامج الدالة عندما أجرى المقارنة بين العددين x و y .

أما في تعريف الدالة نفسها ، فنرى أن الدالة قد تم تعريفها كما لو كانت متغيراً عادياً من النمط الحقيقى . كما أعلن عن نمط البارامترات المستخدمة معها (first,second) أيضاً .

وعند استدعاء الدالة فإن البارامتران first,second يُستبدلان بالبارامترات الحقيقية مثل a,b أو x,y,.... ومن الجائز استخدام بارامترات عددية مع الدالة مثل :

**writeln(max(63.45, 61.23))**

في هذه الحالة فإن نتيجة العبارة تكون هي العدد 63.45 ومن الجائز أيضاً استخدام التعبيرات الحسابية أو الدوال كبارامترات (تماماً كما هو الحال مع الدوال القياسية) مثل :

**writeln(max(sqrt(2), sqrt(3)))**

وهذه العبارة سوف تعطى النتيجة 1.73 .

كما يجوز استخدام الدالة max نفسها كبارامتر وفي هذه الحالة فإنها تستدعى نفسها بنفسها كالمثال الآتي :

**writeln( max(max(6.2, 7.4), max(2.3, 9.5) )**

هذه العبارة سوف يتم تقييمها كما في شكل (٨ - ٢٣) لتنتج العدد 9.5 في النهاية .

$$\begin{array}{c} \max(\underbrace{\max(6.2, 7.4)}_{7.4}, \underbrace{\max(2.3, 9.5)}_{9.5}) \\ \underbrace{\hspace{10em}}_{9.5} \end{array}$$

شكل (٨ - ٢٣)

● بقى أن نتذكر الترتيب المفروض للإعلانات التى يمكن أن يتضمنها أى برنامج ، حيث يجب أن يتبع هذا التسلسل :

CONST	( ١ ) إعلان الثوابت المسماة
TYPE	( ٢ ) إعلان الأنماط المبتكرة
VAR	( ٣ ) إعلان المتغيرات
	( ٤ ) إعلان البرامج الفرعية والدوال المبتكرة .
Procedure	
Function	



## ■ تمارين على الباب الثامن :

س (٨ — ١) اكتب برنامجاً لترتيب ثلاثة أعداد ترتيباً تنازلياً باستخدام المنطق الآتى :

- ١ — رتب العدد الأول والعدد الثانى (وهذا قد يتضمن تبديلهما) .
- ٢ — رتب العدد الأول والثالث (يحتوى العدد الأول الآن على أكبر قيمة) .
- ٣ — رتب العدد الثانى والثالث .

استخدم فى عملية الترتيب برنامجاً فرعياً (وليكن اسمه order) بحيث يقوم بالمقارنات والتبديل ، ويستدعى من البرنامج الرئيسى كالاتى :

**order(first, second);**

**order(first,third);**

**order(second, third)**

س (٨ — ٢) اكتب برنامجاً لتقييم التعبير الآتى :

$$x^5 + y^4 + z^3$$

حيث تُستقبل قيم المتغيرات  $x, y, z$  كمدخلات للبرنامج . يجب أن يتضمن برنامجك الدالة (power) .والتي يمكن استخدامها لرفع عدد حقيقى إلى أس صحيح . فعلى سبيل المثال :

ينتج عن استخدام الدالة power بالبارامترات الآتية بعد :

**power(x,5)**

القيمة  $x^5$  .

س (٨ — ٣) اكتب برنامجاً لفرز الأعداد (sorting) الموضحة فى الجدول رقم

(1) لتحويلها إلى الجدول رقم (2) على أن يتضمن البرنامج البرامج الفرعية الآتية .

- ١ — برنامج لطباعة الجدول رقم (1) .
- ٢ — برنامج لفرز الأعداد بالجدول وترتيبها تصاعدياً (2) .
- ٣ — برنامج لطباعة الجدول رقم (2) (يجوز أن يكون هو نفسه البرنامج الفرعى الأول) .

Array Contents:										
1:	92	91	94	93	96	95	98	97	100	99
11:	82	81	84	83	86	85	88	87	90	89
21:	72	71	74	73	76	75	78	77	80	79
31:	62	61	64	63	66	65	68	67	70	69
41:	52	51	54	53	56	55	58	57	60	59
51:	42	41	44	43	46	45	48	47	50	49
61:	32	31	34	33	36	35	38	37	40	39
71:	22	21	24	23	26	25	28	27	30	29
81:	12	11	14	13	16	15	18	17	20	19
91:	2	1	4	3	6	5	8	7	10	9

الجدول 1

Array Contents:										
1:	1	2	3	4	5	6	7	8	9	10
11:	11	12	13	14	15	16	17	18	19	20
21:	21	22	23	24	25	26	27	28	29	30
31:	31	32	33	34	35	36	37	38	39	40
41:	41	42	43	44	45	46	47	48	49	50
51:	51	52	53	54	55	56	57	58	59	60
61:	61	62	63	64	65	66	67	68	69	70
71:	71	72	73	74	75	76	77	78	79	80
81:	81	82	83	84	85	86	87	88	89	90
91:	91	92	93	94	95	96	97	98	99	100

الجدول 2

شكل (٨ - ٢٤)

س (٨ - ٤) اكتب برنامجاً لطبع الأعداد الأولية (primes) التي تنحصر بين العدد ١ صفر ، والعدد "1000" .

ملاحظة الأعداد الأولية هي التي لا تقبل القسمة إلا على نفسها (انظر الشكل الذي يمثل نتيجة البرنامج) .

Running

2  
3  
5  
7  
11  
13  
17  
19  
23  
29  
31  
37  
41  
43  
47  
53  
59  
61  
67  
71  
73  
79  
.  
.  
.  
> .

الأعداد  
الأولية

شكل (٨ - ٢٥)

# ملاحق الكتاب





## الملحق ( أ ) الكلمات المحجوزة (Reserved Words)

and	function	program
array	goto	record
begin	if	repeat
case	in	set
const	label	then
div	mod	to
do	nil	type
downto	not	until
else	of	var
end	or	while
file	packed	with
for	procedure	

ملاحظة : بعض الكلمات التي وردت هنا لم تتعرض لها في هذا الكتاب .

الملحق (ب) الكلمات القياسية  
Standard Words

**Constants:** false, true, maxint  
**Types:** integer, real, boolean, char, text  
**Files:** input, output  
**Functions:** abs, arctan, chr, cos, eof, eoln, exp, ln, odd, ord, pred, round, sin, sqr, sqrt, succ, trunc  
**Procedures:** dispose, get, new, pack, page, put, read, readln, reset, rewrite, unpack, write, writeln  
**Directive:** forward

ويضيف طراز اللغة UCSD هذه المجموعة أيضاً :

**Types:** string, interactive  
**Functions:** atan, (in place of arctan), log, pwroften, length, pos, concat, copy  
**Procedures:** delete, insert, str, close, seek, mark, release

ملاحظة : بعض الكلمات التي وردت هنا لم تتعرض لها في الكتاب .

## الملحق (ج) الدوال القياسية في لغة باسكال

### PASCAL standard functions

الدالة	نمط الدليل (أو البارامتر)	نمط النتيجة	التعريف
abs	صحيح حقيقي	صحيح حقيقي	القيمة المطلقة
sqr	صحيح حقيقي	صحيح حقيقي	التربيع
sin	صحيح / حقيقي	حقيقي	الجيب (الطبيعي)
cos	صحيح / حقيقي	حقيقي	جيب تمام (الطبيعي)
exp	صحيح / حقيقي	حقيقي	الدالة الأسية $e^x$
Ln	صحيح / حقيقي	حقيقي	اللوغاريتم الطبيعي
sqrt	صحيح / حقيقي	حقيقي	الجذر التربيعي
arctan	صحيح / حقيقي	حقيقي	(ظل <sup>-1</sup> ) الزاوية التي ظلها
odd	صحيح	منطقي	اختبار الأعداد الفردية والزوجية
trunc	حقيقي	صحيح	دالة القطع
round	حقيقي	صحيح	دالة التقريب
ord	لينة	صحيح	
chr	صحيح	لينة	لتحويل العدد الترتيبي إلى اللينة المناظرة
succ	لينة	لينة	لإيجاد اللينة التالية في الترتيب
pred في الترتيب	لينة	لينة	لإيجاد اللينة السابقة
eoln	ملف من اللينات	منطقي	لاختبار نهاية السطر
eof	ملف	منطقي	لاختبار نهاية الملف

ملاحظة (١) :

- في حالة الجيب وجيب التمام يجب أن يكون الدليل بالتقدير الدائري .
- في حالة « الزاوية التي ظلها »  $\arctan$  تكون النتيجة بالتقدير الدائري .

ملاحظة (٢) :

تدل كلمة صحيح/ حقيقي أن دليل الدالة يمكن أن يكون صحيحاً أو حقيقياً أو تركيبة من الأعداد الصحيحة والحقيقية .



## الملحق (د) المؤثرات في لغة باسكال

### Pascal operators

المؤثر	الأولوية	نمط العامل	نمط النتيجة	التعريف
not	1	منطقي	منطقي	النفي المنطقي
★	2	حقيقي / صحيح	حقيقي / صحيح	الضرب
/	2	حقيقي / صحيح	حقيقي	القسمة
div	2	صحيح	صحيح	القسمة الصحيحة
mod	2	صحيح	صحيح	باق القسمة
and	2	منطقي	منطقي	الضرب المنطقي
+	3	حقيقي / صحيح	حقيقي / صحيح	الجمع
-	3	حقيقي / صحيح	حقيقي / صحيح	الطرح
or	3	منطقي	منطقي	الجمع المنطقي
=	4	حقيقي / صحيح	منطقي	التساوي
< >	4	حقيقي / صحيح	منطقي	عدم التساوي
<	4	حقيقي / صحيح	منطقي	أقل من
>	4	حقيقي / صحيح	منطقي	أكبر من
>=	4	حقيقي / صحيح	منطقي	أكبر من أو يساوي
<=	4	حقيقي / صحيح	منطقي	أقل من أو يساوي
in	4	صحيح / لينة	فئة	عضوية الفئة

### ملاحظة :

- تدل كلمة صحيح/حقيقى أن المعاملات التى يؤثر عليها المؤثر يمكن أن تكون صحيحة أو حقيقية أو تركيبة من المعاملات الصحيحة والحقيقية .
- بعض المؤثرات المذكورة يمكن أن تؤثر على معاملات من أنماط أخرى ولكن ذلك لم يناقش فى الكتاب .



**الملحق (هـ)**  
**شرح المؤثرات المنطقية**  
**(logical operators)**

**المؤثر المنطقي AND :**

إذا أردت أن تشغل جهازاً كهربياً مثل التلفزيون فإنك لا بد أن تأخذ إجراءاتهما :

- ١ — وضع القابس (الفيشة) في منبع التيار الكهربى .
- ٢ — فتح الجهاز بواسطة مفتاح القفل والفتح ON/OFF .

فإذا أجريت أحد الخطوتين دون الأخرى لن تتحقق النتيجة المطلوبة وهى تشغيل جهاز التلفزيون .

أى أن شرطين لا بد أن يتحققا معاً لتحقيق النتيجة :

وهذه هى وظيفة المؤثر المنطقي AND فإذا رمزنا للشرط الأول بالمتغير A ، وإذا رمزنا للشرط الثانى بالمتغير B ، فإن التعبير الآتى يعبر عن عملية تشغيل جهاز التلفزيون (أو أى عملية مماثلة) :

**A AND B**

وقيمة هذا التعبير تصبح "TRUE" عندما تأخذ المتغيرات A ، B القيمة TRUE أيضاً . كما يمكن أن يشمل التعبير أكثر من متغيرين . ويمكن تلخيص ذلك بالجدول الآتى :

(لاحظ أن الحرف F هو اختصار كلمة false والحرف T هو اختصار كلمة true) .

<b>A</b> الشرط الأول	<b>B</b> الشرط الثانى	<b>A AND B</b> نتيجة العملية
F	F	F
F	T	F
T	F	F
T	T	T

ويسمى المؤثر AND مؤثر الضرب المنطقى .

**المؤثر المنطقى OR :**

بعض العمليات يمكن تحقيقها بأحد طريقتين مختلفتين . فمثلاً الحجرة التى لها بابان يمكن الخروج منها من أى من البابين . كذلك مصابيح الإضاءة فى بئر السلم يمكن تشغيلها بأى زر من الأزرار (المفاتيح) فى أى دور .  
معنى ذلك أن تحقق أى شرط من الشرطين (أو الشروط) يمكن أن يؤدي إلى النتيجة المطلوبة .

فإذا كان زر النور فى أحد الأدوار هو A وفى دور آخر هو B فإن التعبير الآتى يعبر عن عملية إضاءة نور السلم :

**A OR B**

ويأخذ هذا التعبير القيمة TRUE عندما يكون أى من A أو B يحتوى على القيمة TRUE ويمكن تلخيص ذلك بالجدول الآتى :



<b>A</b> الشرط الأول	<b>B</b> الشرط الثاني	<b>A OR B</b> نتيجة العملية
F	F	F
F	T	T
T	F	T
T	T	T

ويسمى المؤثر OR مؤثر الجمع المنطقي .

(٥-٤-٣) المؤثر المنطقي NOT :

أما المؤثر المنطقي NOT فكما هو ظاهر من اسمه فإنه يقوم بعكس الشروط ، ولذلك فهو يسمى مؤثر النفي المنطقي .

فإذا كان المتغير A يحتوى على القيمة TRUE ، فإن A NOT تصبح قيمته FALSE والعكس بالعكس .

ويمكن التعبير عن ذلك بالجدول الآتي :

<b>A</b>	<b>NOT A</b>
T	F
F	T

الملحق (و)  
حلول التمرينات المختارة

حل تمرينات الباب الأول

جـ (١ - ١) ، جـ (١ - ٢)

(a) جائز

(b) جائز

(c) جائز

(d) غير جائز

(e) جائز

(f) غير جائز (كلمة محجوزة)

(g) غير جائز (المسافة الخالية غير مسموح بها)

(h) جائز ولكن لا يوصى باستخدام كلمات قياسية

(i) جائز ولكن هذا يحرمنا من استخدام كلمة أخرى مثل datafile1 (أكثر من ثمانية حروف متشابهة)

(j) غير جائز لأن الاسم يجب أن يبدأ بحرف كما لا يجوز أن يحتوى على العلامة

(-).

## حل تمارينات الباب الثاني

جـ (٢ - ١) :

البرنامج التالي يحتوى على الإعلان المطلوب في الفقرة الأولى من السؤال ونلاحظ به الآتي :

- درجات الامتحان نعبر عنها بالمتغير العددي الصحيح mark .
- متوسط الدرجات نعبر عنه بالمتغير الحقيقي anverage .
- النتيجة (ناجح/راسب) نعبر عنها بالمتغير المنطقي pass .
- مستوى الامتحان نعبر عنه بمتغير اللينة level .

أما الفقرة الثانية من البرنامج فهي تحتوى على عبارات التخصيص المطلوبة وهي تبدأ بالعبرة BEGIN وتنتهى بالعبرة END .

```
PROGRAM example;
VAR mark      : integer;
    average   : real;
    pass      : boolean;
    level     : char;
BEGIN
    mark := 51;
    average := 47.5;
    level := 'a';
    pass := true;
END.
```

شكل (١)

جـ (٢ - ٢) :

التعبير	قيمة التعبير
$16/5$	3.2
$16 \div 5$	3
$16 \bmod 5$	1
$19/5$	3.8
$19 \div 5$	3
$19 \bmod 5$	4
$8 \div 3 * 3$	6
$7 + 5 \div 3$	8
$13 - 5 \bmod 3$	11

شكل (٢)

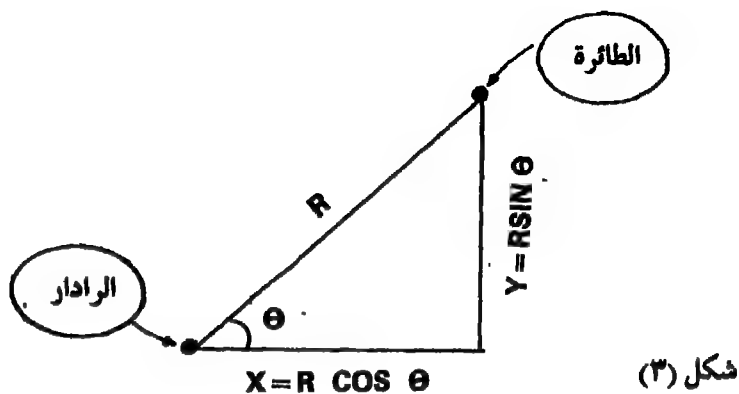
جـ (٢ - ٣) :

لاحظ استخدام المتغيرات الآتية في الإعلان :

degtorad · ثابت مسمّى للتحويل من درجة إلى زاوية نصف قطرية .

- elevation متغير حقيقي لتمثيل زاوية الارتفاع  $\theta$  بالدرجات
- beamrage · متغير حقيقي لتمثيل المدى في اتجاه الاشعاع الرادارى R .
- horizontal متغير حقيقي لتمثيل البعد الأفقى الكارتيزى X .
- angles متغير حقيقي لتمثيل البعد الرأسى الكارتيزى Y .
- radians متغير حقيقي لتمثيل الزاوية  $\theta$  التقديرى النصف قطرى .

أنظر الرسم التالى لتسهيل المسألة من الناحية الفنية :



البرنامج

```

PROGRAM me29 (input, output);
CONST degtorad = 0.0174532925;
VAR elevation,
    beamrange,
    horizontal,
    angels,
    radians    : real;
BEGIN
    read (elevation, beamrange);
    radians := elevation * degtorad;
    horizontal := beamrange * cos (radians);
    angels := beamrange * sin (radians);
    writeln ('bandits at ', horizontal, ' units ');
    writeln ('angels ', angels );
END.
    
```

شكل (٤)

جـ (٢ - ٤) :

```

program journey(input,output);

var dist, mph, starttime, hours, mins : integer;
    realhours : real;

begin

    read(dist, mph, starttime);

    hours    := starttime div 100;
    mins     := starttime mod 100;
    realhours:= hours + mins/60;

    حساب زمن الوصول كعدد حقيقي
    من الساعات بعد منتصف الليل
    ↙
    realhours:= realhours + dist/mph;

    تحويل الزمن إلى ساعات ودقائق
    ↘
    hours    := trunc(realhours);
    mins     := round((realhours - hours)*60);
    write(' arrival time: ', hours, '-', mins)

end.

```

شكل (٥)

## حل تمرينات الباب الثالث

ج (٣ - ١) :

```

program electricitybill2(input,output);

const unitrate = 3.4;
      standcharge = 1.14;
      asterisks = '*****';

var present, previous : integer;

begin
    read(present,previous);
    writeln(asterisks);
    writeln('present meter reading ', present:7);
    writeln('previous meter reading', previous:7);
    writeln('units used
              present - previous :7);
    writeln('rate per unit
              unitrate :6:1, 'p');
    writeln('standing charge      £', standcharge:7:2);
    writeln('sum due is           £',
              (present-previous)*unitrate/100+standcharge:7:2);
    writeln(asterisks)

end.
    
```

شكل (٦)

ج (٣ - ٢)

```

var a,b,c,d:integer;
var av:real;
begin
  read (a,b,c,d);
  writeln;
  av:=(a+b+c+d)/4;
  writeln('marks are: ',a,', ',b,', ',c,', ',d);
  writeln('average is: ',av)
end.

```

البرنامج

التنفيذ

10 20 30 40 ← الأرقام المدخلة  
 marks are: 10,20,30,40  
 average is: 2.5000000000E+01

← النتائج

>

شكل (٧)

جـ (٣ - ٣)



```

program sumandproduct:
var a,b:integer;
begin
  read (a,b);
  writeln;
  writeln(a,'+',b,'=',a+b);
  writeln(a,'*',b,'=',a*b)
end.

```

البرنامج

5 6 ← الأرقام المدخلة  
 5+6=11 ← النتائج  
 5\*6=30  
 >

التنفيذ

شكل (٨)

```

nam payroll (input,output);
hours,payrate,overtime:integer;
gross:real;

```

البرنامج

```

read(hours,payrate,overtime);
writeln;
gross:=hours*payrate+overtime*payrate*1.5;
writeln('gross pay is:', '*',gross)

```

التنفيذ

المخرجات ← 10 10  
 ss pay is:\$ 5.50000000000E+02

إجمالي الأجر

شكل (٩)

ج (٣ - ٨)

بعد إعلان المتغيرات يتم في هذا البرنامج قراءة طول الجسم ساكناً ،  
 بالعبار read .

ونلاحظ أسماء المتغيرات الآتية المستخدمة في البرنامج :

lnew طول الجسم أثناء الحركة .

lrest طول الجسم ساكناً .

v سرعة الجسم

c سرعة الضوء (ثابت مسمى)

```

PROGRAM fitzgerald (input, output);
CONST c = 2.99792458e+8;
VAR lrest,
    lnew,
    v      : real;
BEGIN
    read (lrest);
    read (v);
    lnew := lrest * sqrt
              (1.0 - sqr (v) / sqr (c));
    writeln (lnew)
END.

```

شكل (١٠)



## حل تمارين الباب الرابع

جـ (٤ - ١) :

```
program multiplication(input, output);
var x,i:integer;
begin

  read(x);
  writeln;
  for i:=1 to 10 do
    writeln(i,'x',x,'=',i*x)

end.
```

البرنامج

العدد المدخل

5  
1x5=5  
2x5=10  
3x5=15  
4x5=20  
5x5=25  
6x5=30  
7x5=35  
8x5=40  
9x5=45  
10x5=50

جدول الضرب

العدد 5

التنفيذ

شكل (١١)

جـ (٤ - ٢) :

٣٥٠

```

program seven (input,output);
var n,i:integer;
begin
  read (n);
  writeln;

  for i:=1 to n do
    write('*');

  writeln;

  for i:=1 to n-1 do
    writeln('*':n-i)
end.

```

البرنامج

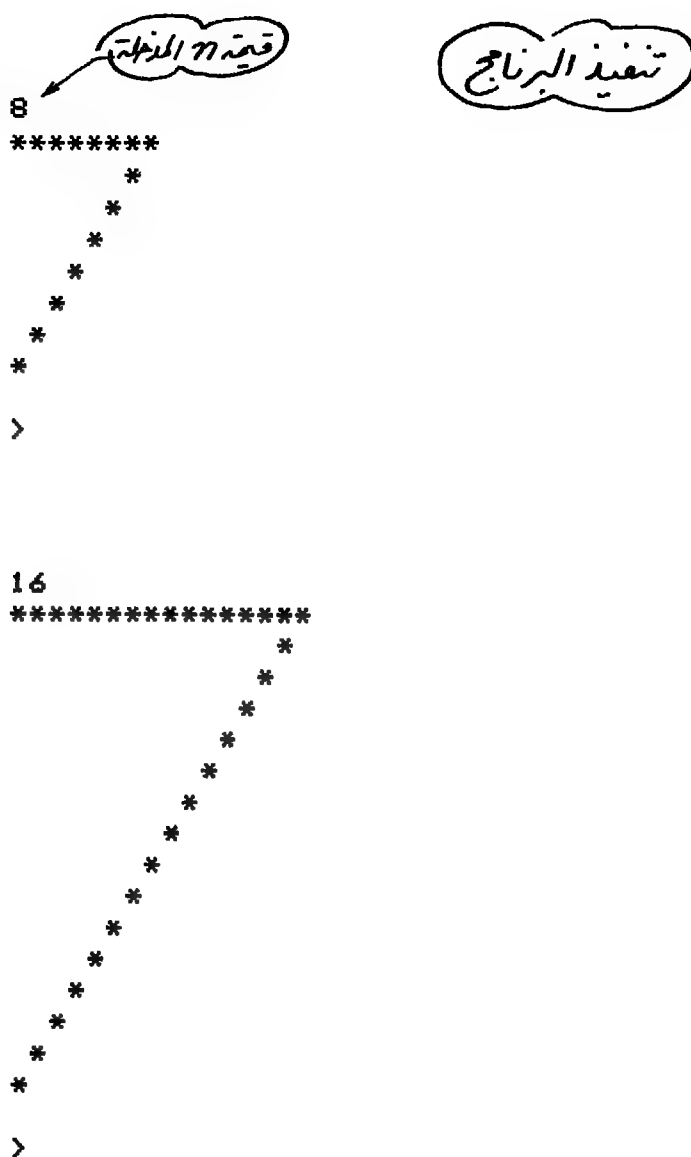
## شكل (١٢)

هذا البرنامج يقوم برسم الرقم 7 بالنجوم التي يتوقف عددها (وبالتالي حجم الرقم 7 على المتغير  $n$ ).

ويبدأ البرنامج بقراءة العدد  $n$  يليه طباعة سطر خال ثم تبدأ حلقة تكرارية لرسم السطر الأول من النجوم وقد استخدمنا فيها العبارة `write` التي تتميز بعدم الانتقال إلى السطر التالي بعد إتمام الطباعة.

وبعد انتهاء هذه الحلقة يتم الانتقال إلى السطر التالي بواسطة العبارة `writeln` يليها الحلقة التكرارية التي ترسم الخط المائل الممثل لجسم الرقم 7 ونلاحظ أن الصيغة المستخدمة للطباعة هي  $(n-i)$  وهي تجعل المسافة التي تظهر عندها النجمة \* تنكمش عند كل سطر تال وهذا هو التنفيذ لقيمتين مختلفتين للمتغير

:  $n$



شكل (١٣)

ج (٤ - ٣) :

```

program triangle (input,output);
var n,i:integer;
begin
  read (n);
  writeln;
  writeln('*':n);

  for i:=1 to n-2 do
    writeln('*':n-1,'*':i*2);

  for i:=1 to 2*n-1 do
    write('*')
  end.

```

البرنامج

8

```

      *
     **
    ***
   ****
  *****
 *****
  *****
   ****
    ***
     **
      *
*****
>

```

التنفيذ لقيم مختلفة للمتغير n

قيمة n المدخلة  
16

```

              *
             **
            ***
           ****
          *****
         *******
        *********
       *********
      *******
     *******
    *******
   *******
  *******
 16 *****
>

```

شكل (١٤)

٣٥٣

: (٦ - ٤) ج

```

program month(input,output);
var m:integer;
begin
  writeln('please enter month number');
  read (m);
  writeln;
  case m of
    1:write(' january');
    2:write(' february');
    3:write(' march');
    4:write(' april');
    5:write(' may');
    6:write(' june');
    7:write(' july');
    8:write(' august');
    9:write(' september');
    10:write(' october');
    11:write(' november');
    12:write(' december');
  end;
  write('.. this month contains: ');
  case m of
    1,3,5,7,8,10,12:writeln('31 days');
    4,6,9,11:writeln('30 days');
    2:writeln('28 or 29 days depending on leap
      years');
  end
end.

```

شكل (١٥)



تنفيذ البرنامج :

```

please enter course number
10
course 10 fri 9am fri 11am
>

```

جـ (٤-٨) :

البرنامج :

```

program month(input,output);
var course:integer;
begin

```

```

    writeln('please enter course number');
    read (course);
    writeln;

```

```

case course of
1,2:write('course ','course','
3: write('course ','course','
4,5:write('course ','course','
6,7:write('course ','course','
8: write('course ','course','
9: write('course ','course','
10: write('course ','course','
endi
end.

```

شكل (١٦)

جـ (٤ - ١) :

```

program election2(input,output);

var party1next, party1overall,
    party2next, party2overall :integer;

begin
    party1overall := 0;
    read(party1next);
    repeat
        party1overall := party1overall + party1next;
        read(party1next)
    until party1next < 0;

    party2overall := 0;
    read(party2next);
    repeat
        party2overall := party2overall + party2next;
        read(party2next)
    until party2next < 0;

    writeln('party1: ', party1overall, '
            'party2: ', party2overall)
end.

```

انتخابات

شكل (١٧)

## حل تمرينات الباب الخامس

```

program salesanalysis(input,output);
var dept, largesales : integer;
    nextitemprice : real;

begin
    for dept := 1 to 63 do
    begin
        largesales := 0;
        read(nextitemprice);

        repeat
            if nextitemprice >= 10 then
                largesales := largesales + 1;

            read(nextitemprice);
        until nextitemprice < 0;

        writeln('dept no. ', dept, ' has made '
                largesales, ' large sales')
    end
end.
    
```



### شكل (١٨)

في هذا البرنامج استخدمت المتغيرات الآتية :

- dept متغير العداد لأقسام الشركة .
- largesales متغير لتجميع السلع التي يزيد سعرها عن 10 جنيهات .
- nextprice متغير حقيقي لتمثيل سعر السلع المختلفة .
- ويتم التمييز بين كل قسم وآخر برقم القسم الموجود في متغير العداد dept .

جـ (٥ - ٢) :

```
program tolerance4(input,output);
```

```
const standard = 6.37;
```

```
var length : real;
```

```
noabove, nobelow, nowithin : integer;
```

```
begin
```

```
nowithin := 0; noabove := 0;
```

```
nobelow := 0; read(length);
```

```
repeat
```

```
if abs(length - standard) < 0.1 then
```

```
nowithin := nowithin + 1
```

```
else
```

```
if length - standard >= 0.1 then
```

```
noabove := noabove + 1
```

```
else
```

```
nobelow := nobelow + 1;
```

```
read(length)
```

```
until length < 0;
```

```
writeln(nowithin, ' values are within tolerance.');
```

```
writeln(noabove, ' values are too large.');
```

```
writeln(nobelow, ' values are too small.')
```

```
end.
```

المراقبة الرقمية للأنتاج

فرز الأطوال غير المقبولة

شكل (١٩)

المتغيرات المستخدمة في البرنامج هي :

( ١ ) ثوابت : standard تمثيل الطول القياسي 6.37 .

( ٢ ) متغيرات حقيقية : length تمثيل الطول .

( ٣ ) متغيرات صحيحة :

noabove عدد الأطوال الأكبر من اللازم .

nobelow عدد الأطوال الأصغر من اللازم .

nowithin عدد الأطوال المقبولة .



## حل تمرينات الباب السادس

ج (٦ - ١) :

```

program codeshift(input,output);
const shift = 5;
var character : char; ordinal : integer;
begin
    repeat
        while not eoln(input) do
            begin
                read(character);
                if character in ['a'..'z'] then
                    begin
                        ordinal := ord(character);
                        ordinal := ordinal + shift;
                        if ordinal > ord('z') then
                            ordinal := ordinal - 26;
                        write(chr(ordinal))
                    end
                else write(character)
            end;
        readln; writeln
    until eof(input)
end.

```

شكل (٢٠)

: (٣ - ٦) ↗

```

program countwords(input, output);
var nextch : char;
    noofletters, noofwords, noof4words : integer;
begin
    noofwords := 0; noof4words := 0;
    repeat
        repeat read(nextch) until nextch in ['a'..'z'];

        noofletters := 0;
        repeat
            noofletters := noofletters + 1;
            read(nextch)
        until not (nextch in ['a'..'z']);

        noofwords := noofwords + 1;
        if noofletters = 4 then noof4words := noof4words
        until nextch = '.';

        writeln('words in sentence: ', noofwords);
        writeln('4-letter words in sentence: ', noof4words)
    end.

```

شکل (٢١)

## حل تمارين الباب السابع

ج (٧ - ١) :

```

program chararrays(input,output):
VAR ch: array[1..20] of char;
    n,i: integer;

BEGIN
    readln(n); ← قراءة عدد الحروف
    for i:=1 to n do ← قراءة الحروف
        read(ch[i]);

        writeln; ← سطر خالي
    for i:=1 to n do ← طباعة الحروف
        writeln(ch[i]);
end.
    
```

شكل (٢٢)



```

program salesanalysis4(input,output);  : (٧ - ٢) ج
var salesfor : array [1..8] of integer;
    dept, col, scaledtotal : integer;

begin
    for dept := 1 to 8 do
        salesfor[dept] := 0;

        read(dept);
        repeat
            salesfor[dept] := salesfor[dept] + 1;
            read(dept);
        until dept = -1;

        for dept := 1 to 8 do
            begin
                scaledtotal := salesfor[dept] div 10;
                for col := 1 to scaledtotal do
                    write('*');
                writeln
            end;
        end.
    
```

شكل (٢٣)

شكل (٢٤) : ج (٧ - ٣)

```

program popmap2(input,output);
var popmap : array [1..8,1..10] of integer;
    row, col, rowa, cola, rowb, colb, subtotal; integer;

begin
    for row := 1 to 8 do
        for col := 1 to 10 do
            read(popmap[row,col]);

            subtotal := 0;
            read(rowa, rowb, cola, colb);

            for row := rowa to rowb do
                for col := cola to colb do
                    subtotal := subtotal + popmap[row,col];

                writeln('population of sub-zone is ', subtotal)
            end.
    
```

جميع عدد سكان المنطقة الجزئية

ج (٧ - ٦)

```

program times(input,output);

var day, period, room, lecture : integer;
    place : array [1..5,1..6] of integer;

begin
    تأخير المتغيرات بقيمة ابتدائية (صفر)
    for day := 1 to 5 do
        for period := 1 to 6 do
            place[day,period] := 0;

        إدخال البيانات
        for lecture := 1 to 20 do
            begin
                read(day,period,room);
                place[day,period] := room;
            end;

            الطباعة
            writeln('period':28);
            writeln('1 2 3 4 5 6':34);
            writeln('-----':34);

            for day := 1 to 5 do
                begin
                    case day of
                        1 : write('mon');
                        2 : write('tue');
                        3 : write('wed');
                        4 : write('thu');
                        5 : write('fri')
                    end;

                    write(' ');
                    for period := 1 to 6 do
                        if place[day,period] = 0 then
                            write(' ')
                        else
                            write(place[day,period]:3);
                        end;
                    end;
                    writeln
                end
            end
        end
    end
end.

```

جدول المحاضرات

شكل (٢٥)

ج (٧ - ٨)

```

program terms:
  type term=(first.second.summer):
  var thisterm,nextterm: term;

begin
  {نقطة انه قيمة ما قد اعطيت للمتغير thisterm}
  if thisterm=summer
  then
    nextterm:=first
  else
    nextterm:=succ(thisterm);
end.

```

شكل (٢٦)

ج (٧ - ٩) :

```

PROGRAM matrices (input, output);
CONST   rowlim = 3;
        collim = 5;

TYPE    matrix = ARRAY [1..rowlim] OF
                                ARRAY[1..collim] OF real;
(* or   ARRAY [1..rowlim, 1..collim] OF real; *)
VAR     a, b, c : matrix;
        row      : 1..rowlim;
        col      : 1..collim;

BEGIN
  (* assuming values given to a and b *)
  FOR row:= 1 TO rowlim DO
    FOR col:=1 TO collim DO
      c[row,col] := a[row,col] + b[row,col]
    END.
  END.

```

شكل (٢٧)

ج (٧ - ١٠) :

```

PROGRAM frequencies (input, output);
TYPE  alfachar = 'a'..'z';
VAR    ch : char;
        index: alfachar;
        table: ARRAY [alfachar] OF integer;
BEGIN
    FOR index := 'a' TO 'z' DO table[index] := 0;
    WHILE NOT eof DO
        BEGIN
            read(ch);

            IF (ch >= 'a') AND (ch <= 'z')
                THEN table[ch] := table[ch] + 1
            END;
        FOR index := 'a' TO 'z' DO
            writeln ( index, ' occurs ',
                    table[index], '
            times ' )
        ..END..

```

شكل (٢٨)



## إجابات تمارين الباب الثامن

جـ (٨ - ١) :

```
program sort3numbers(input,output);
var first, second, third : integer;
```

```
    procedure order(var a,b : integer);
    var temp : integer;

    begin
        if a < b then
            begin
                temp := a;
                a := b;
                b := temp;
            end
        end;
    end;
```

البرنامج الفرعي

```
begin
    read(first, second, third);
    order(first, second);
    order(first, third);
    order(second, third);
    writeln(first:6, second:6, third:6)
end.
```

البرنامج الرئيسي

شكل (٢٩)

فرز مصفوفة أعداد

ج (٨ - ٣) :

```

VAR
  i, j, k: INTEGER;
  tab: ARRAY [1..100] OF INTEGER;

PROCEDURE printarr;
{procedure to write out the array}
    طابعة الجدول

BEGIN
  WRITELN; WRITELN (' Array Contents:');
  FOR i := 0 to 9 DO
    BEGIN
      WRITE (10*i+1:5, ':');
      FOR j := 1 TO 10 DO WRITE (tab[10*i+j]:5);
      WRITELN;
    END;
  END;

PROCEDURE sortarr;
{ Procedure to sort the array }
    فرز الأعداد

VAR
  temp: INTEGER;  swap: BOOLEAN;

BEGIN
  REPEAT
    swap := FALSE;
    FOR i := 1 to 99 DO
      IF tab[i] > tab[i+1] THEN
        BEGIN
          temp := tab[i];
          tab[i] := tab[i+1];
          tab[i+1] := temp;
          swap := TRUE;
        END;
    UNTIL NOT swap;
  END;

  BEGIN
    { Initialize the table that will be sorted }
    k := 0;
    FOR i := 9 DOWNT0 0 DO
      FOR j := 1*10+1 TO (i+1)*10 DO
        BEGIN
          k := SUCC(k);
          IF ODD(j) THEN tab[k] := j+1 ELSE tab[k] := j-1;
        END;
      END;

    printarr; طابعة الجدول
    sortarr;
    printarr; طابعة الجدول
  END.

```

شكل (٣٠)

ج (٨ - ٤)

```

var
  prime:      integer;
  rprime:     real;
  i:          integer;
  sqrtp:      integer;
  notprime:   boolean;

begin
  writeln('      2');
  writeln('      3');
  prime := 5;
  repeat
    rprime := prime;
    sqrtp := trunc(sqrt(rprime) + 1.0);
    i := 1;
    notprime := false;
    while (i < sqrtp) and (not notprime) do
      begin
        i := i + 2;
        notprime := (prime mod i = 0);
      end;
    if (not notprime) then writeln(prime:6);
    prime := prime + 2;
  until (prime > 10000);
end.

```

توليد وطبع الأعداد  
الأولية حتى 1000

شكل (٣١)

## وإلى اللقاء دائماً ...

رغم جولتنا الطويلة بين عبارات ودوال وتعبيرات اللغة ، فما زال في لغة باسكال الكثير ، لاسيما أن ما جدّ عليها من تطوير وإضافات جعلها أضخم من أن يضمّها كتاب واحد .

لكننا حاولنا في هذا اللقاء مع قارئنا العزيز أن نقدم له نخبة نافعة من أدوات ووسائل اللغة تمكّنه من بناء برنامج جيد متكامل الإنشاء وتغنيه عن كتاب آخر في وقت قريب . أما الأجزاء التي أغفلناها فهي إما إضافات غير أساسية مثل عبارات الموسيقى وبعض عبارات الرسم ، وإما موضوعات اختلفت فيها الطرازات وأصبح من الضروري الرجوع لكتاب اللغة الخاص بالكمبيوتر المعين ، وهذا ينطبق على موضوع الملفات .

أما موضوع كتابنا القادم عن لغة باسكال بإذن الله ، فسوف يخدم فنون البرمجة أكثر مما يخدم مفردات اللغة وقواعدها . فالمهارة في استخدام أدوات اللغة تجعل المبرمج يصل إلى الهدف المنشود من أقصر الطرق .

أما اللقاء القادم فلعله يكون حول أحد موضوعات الساعة في مجال الكمبيوتر لا سيما وأنا على وشك أن ندخل عصراً جديداً من عصور الكمبيوتر وهو عصر الكمبيوتر ذى الخلايا العصبية !  
وإلى اللقاء دائماً ...

أسامة الحسيني



## المراجع

**Computer Programming in Pascal** — ١

DAVID LIGHTFOOT (Teach yourself)

**Simple Pascal** — ٢

JAMES J MCGREGOR,  
ALAN H WATT (PITMAN)

**Proverbs for Programming in Pascal** — ٣

LOUISE E. MOSER,  
ANDREW A. TURNBULL (Wiley)

# كتب للمؤلف في مجال الكمبيوتر

صدرت عن مكتبة ابن سينا

١ - تحدث مع الكمبيوتر بلغة كوبول

... المستوى الأول

٢ - كل شيء عن الكمبيوتر (وكتابة البرامج بلغة بيسك)

... مبسط للنشء ولأولياء الأمور .

٣ - تحدث إلى الكمبيوتر بلغة بيسك

... حتى المستوى المتقدم من لغة بيسك

يضم اللغة القياسية قديمها وحديثها وأيضاً

أشهر طرازات لغة بيسك .

٤ - كيف يفكر الكمبيوتر

... خرائط التسلسل المنطقي للبرامج والنظم

الآلية وتحويل النظم اليدوية إلى آلية .

٥ - برمجة الألعاب الكمبيوترية

... طرق برمجة القذائف والتصادم والمؤثرات

الصوتية مشروحة بلغة الطرازات الشهيرة

للكمبيوتر المنزلي في مصر والعالم العربي

علاوة على لغة بيسك القياسية

(ميكروسوفت) .

٦ - مدخلك إلى عالم الكمبيوتر (المقدمة الأساسية لعلوم الكمبيوتر)

٧ - تعلم لغة الكمبيوتر من خلال لغة بيسك

... مدخل مناسب للهواة والمحترفين لإيجاد

لغة سي .

## ٨ — الرسم بالكمبيوتر

... يتناول كل ما يخص استخدام الكمبيوتر  
في الرسم .. يشرح عبارات بيسك القياسية  
للرسم الدقيق علاوة على أهم اللهجات  
المنتشرة للأجهزة الكمبيوتر المنزلي .

## ٩ — تحدث إلى الكمبيوتر بلغة لوجو

... لغة أصدقاء الروبوت ..  
مع تطبيقات مختلفة في البرمجة والألعاب  
باستخدام السلحفاة الشهيرة للرسم على  
الشاشة (turtle graphics) المدخل  
المناسب للأطفال إلى عالم الكمبيوتر .

## ١٠ — تحدث إلى الكمبيوتر بلغة فورتران ٧٧

... مرجعك العربي في لغة فورتران يبدأ من  
البدايات الأولى للغة ويصل حتى مستويات  
متقدمة في إنشاء البرامج . يضم الكتاب كل  
عبارات اللغة قديمها وحديثها مع تطبيقات  
على مختلف أجهزة الكمبيوتر .

## ١١ — برامج وألعاب كومبيوترية مشروحة (بلغة بيسك)

... برامج تعليمية فوازير ألعاب حروب  
وقذائف ومغامرات .. علاوة على برامج  
الملفات ومعالجة الكلمات بلغة بيسك . على  
أشهر طرازات الكمبيوتر المنزلي :  
تكساس ، كومودور ، أتاري ، BBC ،  
إليكترون ، سنكلير ..

## ١٢ — قبل أن تشتري كومبيوتر

... دليلك في شراء جهاز كومبيوتر لمنزلك  
أو مكتبك أو محلّك التجارى . دليلك في  
التدريب إذا أردت العمل في أحد مجالات  
الكومبيوتر نقد وتحليل خصائص أجهزة  
الكومبيوتر الشخصية والمنزلية .

وفي مجال الهندسة الكهربائية :

— كل شيء عن الإلكترونيات .

وفي مجال قصص الخيال العلمى للشباب :

— إعدام إنسان آلى .

— الدخول في الثقب الأسود .

— الدخول في الثقب اللأسود .

— المعلوم والمجهول .



## منشورات للمؤلف صدرت في الولايات المتحدة الأمريكية بالاشتراك مع آخرين

(١) تصميم وسيلة كومبيوترية لمعاونة المعوقين تعمل بالتعرف على  
الصوت البشرى باستخدام دوائر الخلايا العصبية .  
بالاشتراك مع :

الأستاذة الدكتورة / جين ميردوك  
الأستاذ الدكتور / عبد الفتاح الحسينى  
(كبير مهندسى شركة تكنولوجيا انترناشيونال)  
المهندس / رودريجو رودريجس  
(مهندس بشركة تكنولوجيا انترناشيونال)

الناشر : IEEE

المؤتمر السنوى لرابطة مهندسى الكهرباء والإلكترونيات — كاليفورنيا  
. ١٩٨٨

J.Y. Murdock, A.A. Hussein, E. Liang, O.A. Hussein, R.J.  
Rodriguez, Improvement on Speech Recognition and Synthesis  
for Disabled Individuals Using Fuzzy Neural Net Retrofits,  
IEEE Annual International Conference on Neural Networks, San  
Diego, CA (1988).

## (٢) محادثة الكمبيوتر بالصوف في أغراض التعليم .

بالاشتراك مع :

الأستاذة الدكتورة / جين ميردوك

الدكتور / إنجو ليانج (مهندس بشركة تكنولوجيا اترناشيونال)

الأستاذ الدكتور / عبد الفتاح الحسيني

الناشر :

المؤتمر السنوى الأول لجمعية دوائر الخلايا العصبية

(Neural Network Society)

بوسطن ، ماساتشوستس ١٩٨٨

J.Y. Murdock, O.A. Husseiny, E. Liang, A.A. Husseiny, A High Confidence Voice Interactive Hybrid Neural System for Learning, First Annual Meeting, International Network Society, Boston, MA (1988)

## (٣) التحكم في الربوط للعمل في المناطق الخطرة

O.A. Husseiny, R.J. Rodriguez, E. Liang, A.A. Husseiny, Programmable Controller for Robotic Systems in Hostile Environment, Nuclear Technology, (1988).

بالاشتراك مع :

- ١ — المهندس / رودريجو رودريجس
- ٢ — الدكتور / إنجو ليانج
- ٣ — الأستاذ الدكتور / عبد الفتاح الحسينى

الناشر :

مجلة الصناعة النووية ١٩٨٨ Nuclear Technology

(٤) التحكم فى الروبوتات الصناعية

بالاشتراك مع :

- ١ — الدكتور / إنجو ليانج
- ٢ — الأستاذ الدكتور / عبد الفتاح الحسينى
- ٣ — المهندس / رودريجو رودريجس

الناشر :

مجلة « الصناعة باستخدام الروبوتات والكمبيوتر » ١٩٨٨

Robotics & computer - Integrated Manufacturing

E. Liang, A.A. Hussein, R.J. Rodriguez, O.A. Hussein,  
Supervisory Controller for Robotics in Manufacturing,  
Robotics and Computer-Integrated Manufacturing, (1988).

(٥) تصميم وسيلة تعليمية للمعوقين تعمل بالتعرف على الصوت  
البشرى .

### بالاشتراك مع :

- ١ — المهندس / ميكيل بارنت
- ٢ — الأستاذة الدكتورة / جين ميردوك
- ٣ — الأستاذ الدكتور / عبد الفتاح الحسينى

### الناشر :

تقرير وزارة التعليم بالولايات المتحدة ١٩٨٨ .

O.A. Hussein, Michael Barnett, contributions to Jane Y. Murdock,  
A.A. Hussein, A User-Friendly Voice Interactive Learning  
Aid for Individuals with Handicaps (VILAH), TII Rep. #  
TILA/7088100/R and DOE/870088-6 (1988).

### (٦) التحكم بالصوت والصورة فى معامل اختبار الروبوتيات

### بالاشتراك مع :

- ١ — المهندس / ميكيل بارنت
- ٢ — المهندس / ريتشارد جارك
- ٣ — الأستاذة الدكتورة / زينب صبرى
- ٤ — المهندس / راندى مانيجولت
- ٥ — المهندس / جون بوش

### الناشر :

تقرير شركة تكنولوجيا انترناشيونال ١٩٨٨ .



O.A. Hussein, contributions to Richard E. Jarka, Zeinab A. Sabri, S. Keith Adams, A.A. Hussein, Randy Manigault, John Bush, Development of Requirements for an Advanced Robotic Laboratory Equipment, TII RD&E Center Technical Report # 12680 (1988).

(٧) تلافى الأخطاء عن طريق جهاز التحكم في الربوط  
بالاشتراك مع :

١ — الأستاذ الدكتور / عبد الفتاح الحسينى

٢ — الدكتور / إنجو ليانج

٣ — المهندس / رودريجو رودريجس

الناشر :

مجلة « روبوطيكا » ١٩٨٨

A.A. Hussein, E. Liang, R.J. Rodriguez, O.A. Hussein, Fault-Tolerant Robotic Controller, Robotica, (1988).

## فهرست الكتاب

٥ ..... مقدمة المؤلف

### ■ الباب الأول : القواعد العامة للغة باسكال

٩ ..... ● مفتتح

١٠ ..... (١-١) لغة باسكال

١٠ ..... (٢-١) ترجمة اللغات عالية المستوى

١١ ..... (٣-١) ترجمة لغة باسكال

١٤ ..... (٤-١) نظرة شاملة إلى برنامج باسكال

٢٣ ..... (٥-١) أسماء البيانات (identifiers)

٢٤ ..... (٦-١) الكلمات المحجوزة (reserved words)

٢٤ ..... (٧-١) الكلمات القياسية (standard words)

٢٤ ..... (٨-١) خرائط قواعد اللغة (syntax diagrams)

٢٥ ..... (٩-١) التعليقات (remarks)

٢٧ ..... ● تمرينات

### ■ الباب الثاني : الأنماط والتعبيرات (Types and expressions)

٣١ ..... ● مفتتح

٣٢ ..... (١-٢) الأنماط القياسية (standard types)

٣٣ ..... (٢-٢) الثوابت (constants)

٣٥ ..... (٣-٢) الإعلان (declaration)

٤٠ ..... (٤-٢) التخصيص (assignment)

٤٣ ..... (٥-٢) التعبيرات والمؤثرات (expressions and operators)

٤٤ ..... (١-٥-٢) التعبيرات الحسابية (arithmetic expressions)

٤٩ ..... (٢-٥-٢) الدوال القياسية (standard functions)

٥٣ ..... (٣-٥-٢) المؤثرات الأسية (exponential operators)

٥٦	..... (Boolean expressions' التعبيرات المنطقية (٢-٦)
٥٦	..... (relational operators) المؤثرات العلاقية (٢-٦-١)
٦٢	..... (boolean operators) المؤثرات المنطقية (٢-٦-٢)
٦٥	..... (De Morgan's) قواعد دي مورجان (٢-٦-٣)
٦٥	..... odd الدالة (٢-٦-٤)
٦٦	..... مقارنة اللبئات (٢-٦-٥)
٦٨	..... تمرينات

## ■ **الباب الثالث : إدخال البيانات — طباعة المعلومات** **(Input-Output)**

٧٣	..... مفتتح
٧٥	..... Read إدخال البيانات إلى البرنامج (٣-١)
٧٨	..... طباعة رسالة عند إدخال البيانات (٣-٢)
٨٠	..... writeln طباعة الرسائل والنتائج (٣-٣)
٨٣	..... write عبارة الطباعة (٣-٤)
٨٥	..... استخدام عبارة الطبع في الرسم (٣-٥)
٨٨	..... (formatted output) صياغة الخرج (٣-٦)
٩٢	..... تمرينات

## ■ **الباب الرابع : التفريع والتكرار (Branching & Looping)**

٩٧	..... مفتتح
٩٨	..... (control statements) عبارات التحكم (٤-١)
٩٨	..... (for-statement) الحلقات التكرارية (٤-٢)
١٠٣	..... for خصائص الحلقة التكرارية (٤-٣)
١٠٤	..... الحلقة التكرارية بخطوة سالبة (٤-٤)
١٠٧	..... (if-statement) الاختيار بين البدائل (٤-٥)
١١٣	..... (٤-٦) تطبيقات على العبارة الشرطية

١٢٠	.....	٤-٧) العبارة الشرطية متعددة النتائج
١٢٢	.....	٤-٨) الاختيار بين البدائل المتعددة : case
١٣٤	.....	٤-٩) الحلقات التكرارية المشروطة (Conditional loops)
١٣٥	.....	٤-٩-١) عبارة التكرار repeat ...
١٤١	.....	٤-٩-٢) عبارة التكرار while
١٤٦	.....	٤-٩-٣) مقارنة بين الحلقتين while, repeat
١٤٧	.....	٤-١٠) طرق مختلفة لمعالجة البيانات بالحلقات التكرارية
١٥٧	.....	● تمارين

## ■ الباب الخامس : منشآت التحكم (control structures)

١٦٥	.....	● مفتاح
١٦٦	.....	٥-١) الحلقات التكرارية المحتوية على عبارات شرطية
١٦٩	.....	٥-٢) الحلقات التكرارية المتداخلة (nested loops)
١٧٩	.....	٥-٣) العبارات الشرطية المتداخلة (nested-if)
١٨٧	.....	٥-٤) ملاحظات على بناء العبارات الشرطية المتداخلة
١٩١	.....	٥-٥) مراجعة البيانات (data validation)
١٩٤	.....	٥-٦) حماية البرنامج من البيانات الخاطئة
١٩٩	.....	● تمارين

## ■ الباب السادس : التعامل مع اللبئات (character manipulation)

٢٠٣	.....	● مفتاح
٢٠٤	.....	٦-١) متغيرات اللبئات (character variables)
٢٠٧	.....	٦-٢) نهاية السطر (eol)
٢٠٩	.....	٦-٣) نهاية الملف (eof)
٢١١	.....	٦-٤) ترتيب اللبئات
٢١٥	.....	٦-٥) المزيد من الوسائل لمعالجة اللبئات
٢١٥	.....	٦-٥-١) طباعة اللبئات على الشاشة

- ٢١٧ ..... pred, succ, chr, ord دوال اللبثات (٢-٥-٦)  
 ٢٢٣ ..... تمرينات ●

### ■ الباب السابع : المصفوفات (arrays)

- ٢٢٧ ..... مفتاح ●  
 ٢٢٩ ..... (Subscripted variables) المتغيرات الدليلية  
 ٢٣١ ..... الإعلان عن المصفوفات (٢-٧)  
 ٢٣٤ ..... معالجة المصفوفات ذات البعد الواحد (٣-٧)

#### (one-dimensional arrays)

- ٢٤٣ ..... الرسم البياني (histograms) (٤-٧)  
 ٢٤٧ ..... شحن المتغيرات بقيمة ابتدائية (initialization) (٥-٧)  
 ٢٤٩ ..... معالجة المصفوفات ذات البعدين (٦-٧)

#### (two-dimensional arrays)

- ٢٥٥ ..... ملاحظات على المتغيرات الدليلية (٧-٧)  
 ٢٥٧ ..... الأنماط المتكررة type (٨-٧)  
 ٢٥٨ ..... أنماط القعات الجزئية (subranges) (١-٨-٧)  
 ٢٥٩ ..... أنماط القوائم (enumeration type) (٢-٨-٧)  
 ٢٦٣ ..... العبارة type (٣-٨-٧)  
 ٢٦٥ ..... أدلة المصفوفات (٩-٧)  
 ٢٧٠ ..... المصفوفات المُحزَمة (packed array) (١٠-٧)  
 ٢٧٣ ..... الحرفيات في طرازات باسكال string (١١-٧)  
 ٢٧٤ ..... دوال الحرفيات string functions (١٢-٧)  
 ٢٨٦ ..... تمرينات ●

### ■ الباب الثامن : البرامج الفرعية والدوال

#### procedures & functions

- ٢٩٣ ..... مفتاح ●

٢٩٥	..... (٨-١) البرنامج الفرعي والبرنامج الرئيسي
٢٩٩	..... (٨-٢) البارامترات (parameters)
٣٠٢	..... (٨-٣) استخدام أكثر من بارامتر
٣١٢	..... (٨-٤) البارامتر المتغير (variable parameter)
٣١٧	..... (٨-٥) البارامترات الدلالية (array parameters)
٣٢٢	..... (٨-٦) الدوال (functions)
٣٢٦	..... ● تمارينات

## ■ ملاحق الكتاب

٣٣١	..... (أ) الملحق (المحجوزة) (Reserved Words) ●
٣٣٢	..... (ب) الكلمات القياسية (Standard Words)
٣٣٣	..... (ج) الدوال القياسية (Standard functions)
٣٣٥	..... (د) المؤثرات (Operators)
٣٣٧	..... (هـ) شرح المؤثرات المنطقية (logical operators)
٣٤٠	..... (و) حلول التمارينات المختارة
٣٧٠	..... ● وإلى اللقاء دائماً
٣٧١	..... ● المراجع
٣٧٢	..... ● كتب للمؤلف صدرت عن دار ابن سينا
٣٧٥	..... ● منشورات للمؤلف صدرت في الولايات المتحدة الأمريكية
٣٨٠	..... ● الفهرست

رقم الإيداع ٧٦٠٠ / ٨٨

دار الناصر للطباعة والإستلامية  
٥ - شارع منشأط شبرا القمامة  
٧٧٣٩٣١ ت



## كلمة غلاف المؤخرة

هذا الكتاب ....

هذا الكتاب ... يرشدك إلى الطريق .

....

فهو يبدأ من أساسيات لغة باسكال ويأخذ بيدك  
في سلسلة إلى بناء البرامج المتكاملة الإنشاء .

....

والكتاب يحتوي على عدد كبير من الأمثلة  
والتجربات المحولة فضلاً عن البرامج التطبيقية في  
مختلف المجالات الرياضية والتجارية ومعالجة  
الكلمات

....

لذلك فالكتاب مُوجه لمتختلف مستويات الدارسين  
كمراجع عربي للغة باسكال وللقرءاء على اختلاف  
تخصصاتهم واهتماماتهم .

Bibliotheca Alexandrina



0220995